Oracle Solaris System Administration Series

Oracle® Solaris Cluster Essentials

ORACLE
SOLARIS

Tim Read

Library of Congress Cataloging-in-Publication Data

Read, Tim, 1963-
Oracle Solaris Cluster essentials / Tim Read.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-13-248622-4 (pbk. : alk. paper)
1. Oracle Solaris Cluster. 2. Integrated software. I. Title.
QA76.76.I57R43 2011
005.5—dc22

2010027191

ISBN-13: 978-0-13-248622-4
ISBN-10: 0-13-248622-9
Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.
First printing, September 2010

Contents

Preface

It has now been over eight years since Richard Elling and I wrote our Sun BluePrints *Designing Enterprise Solutions with Sun Cluster 3.0* [Design] to herald the release of the new Sun Cluster 3.0 software. Since then, the Solaris Cluster software has grown and matured, providing a stable and robust clustering product on which you can confidently run your business- and mission-critical services.

Now with the completion of the Oracle acquisition, the Oracle Solaris Cluster software is part of a much larger software portfolio. This software will continue to provide the best-of-breed high-availability solutions for the Oracle Solaris platform, addressing business continuity and disaster recovery for customers deploying Oracle and non-Oracle databases and applications.

Much of the additional content for this book is drawn from white papers and Sun BluePrints documents that I have authored or coauthored over the intervening years, augmented with material taken from Sun blog postings, documentation, and other Sun sources of technical information. See the complete list of sources used for this book in the Bibliography at the end of the book.

Scope

This book is written from the perspective that the Solaris 10 10/09 release is your default operating system and that you have installed, or intend to install, the Solaris Cluster 3.2 11/09 software. Because this book represents the Solaris Cluster software capabilities at the time of this writing, I have deliberately not included

many product configuration restrictions. Inevitably, some of these restrictions might be lifted as the product is tested and developed. Consequently, always check the latest product documentation, or contact Oracle technical staff, if you have any questions about a particular configuration or a product feature.

Although the book provides considerable detail on both Oracle Solaris Cluster and Oracle Solaris Cluster Geographic Edition, I have ultimately had to draw the line regarding what content to include and what content to omit. In general, I have not gone in depth into any of the Solaris Cluster agents or specific command-line options. My expectation is that this book gives you a sufficiently detailed description of the products' features and capabilities to provide a solid grounding in the subject matter. From there, you can expand your knowledge through reading the extensive product literature.

The book covers both the Oracle Solaris Cluster and Oracle Solaris Cluster Geographic Edition products. Because the Solaris Cluster software is a prerequisite for the Solaris Cluster Geographic Edition software, the book covers the Solaris Cluster software first. Chapters 1 through 4 are devoted to the Solaris Cluster software, and Chapters 5 through 7 describe the Solaris Cluster Geographic Edition software. The last two chapters, 8 and 9, provide several detailed examples of how to use the respective products.

Although every effort has been made to check the accuracy of the information in this book, it is almost inevitable that some errors have evaded the scrutiny of the book's reviewers or my corrections based on their feedback. So, I think the following quotation from one of my favorite books of all time is pertinent:

> *The Hitch Hiker's Guide to the Galaxy* is an indispensable companion to all those who are keen to make sense of life in an infinitely complex and confusing Universe, for though it cannot hope to be useful or informative on all matters, it does at least make the reassuring claim, that where it is inaccurate it is at least *definitively* inaccurate. In cases of major discrepancy it is always reality that's got it wrong.[1]

Intended Audience

This book is intended to serve as a valuable resource for three broad categories of readers. These readers span the entire lifecycle of a cluster system, from initial design through implementation and finally ongoing support.

- *Data center architects* will benefit from the material that describes the product's requirements, features, and capabilities. This book enables them to

1. Douglas Adams, *The Restaurant at the End of the Universe* (Pan paperback edition), p. 35.

design Solaris Cluster systems suited to providing the levels of availability required by the target applications.

- *System implementers* will derive the most value from the sections that describe how the Solaris Cluster resource group and resource constructs can best be used to encapsulate the target applications.
- *System administrators* will find the book a useful source of background material on how the Solaris Cluster features work. This material as well as the sections on troubleshooting will help them resolve day-to-day issues with the cluster. In addition, the sections on cluster management will provide them with guidance on maintaining a Solaris Cluster system after it has been moved into production.

The book assumes that the reader is familiar with the Oracle Solaris operating system (Oracle Solaris OS).

How to Use This Book

Readers who just want to know about the high-availability features provided by the Solaris Cluster software, and who are not interested in disaster recovery, will benefit from reading the first four chapters of the book. Chapter 1 focuses on the basic requirements for designing a Solaris Cluster system, and Chapter 2 provides a more in-depth description of the Solaris Cluster features. If you are considering implementing virtualization technologies, Chapter 3 explains how the Solaris Cluster software can coexist with these environments. Finally, Chapter 4 discusses some of the tasks that you perform to manage your Solaris Cluster system.

Readers who want to understand the capabilities of the Solaris Cluster Geographic Edition software, or who are planning a disaster recovery system and are already familiar with the Solaris Cluster software, should start with Chapter 5. In many ways, Chapter 5 mirrors Chapter 1 insofar as it provides a general background to the Solaris Cluster Geographic Edition product. More detailed technical information about the features of the Geographic Edition product is explained in Chapter 6. Finally, Chapter 7 covers some additional tasks that you might need to perform while managing a Solaris Cluster Geographic Edition installation.

If you learn best by playing with a piece of software and then reading the associated background material, then Chapter 8 and Chapter 9 are the places to start. Each chapter has detailed examples that you can follow to create your own Solaris Cluster or Solaris Cluster Geographic Edition installation. Bear in mind that these examples are not necessarily optimal configurations or what might be considered "best practice." Instead, the examples focus on demonstrating feature capabilities to give you a platform on which you can build and experiment.

Note

Oracle Corporation acquired Sun Microsystems, Inc., early in 2010, when this book was nearing completion. Although this book mostly uses the new product names, occasional reference is also made to previous names. The following table provides a guide to the old and new product names.

Sun Product Name	Oracle Product Name
Solaris	Oracle Solaris
Solaris Cluster	Oracle Solaris Cluster
Solaris Cluster Geographic Edition	Oracle Solaris Cluster Geographic Edition
Solaris Zones or Solaris Containers	Oracle Solaris Zones or Oracle Solaris Containers
Logical Domains	Oracle VM Server for SPARC
VirtualBox	Oracle VM VirtualBox
ZFS	Oracle Solaris ZFS

4

Managing Your Oracle Solaris Cluster Environment

This chapter covers many important aspects of an Oracle Solaris Cluster system's lifecycle. It begins by outlining some of the planning you need to do prior to installing the Oracle Solaris operating system on a cluster node. Next comes information about how to secure your newly installed system against attack. The options for installing the Solaris Cluster software are then covered, followed by information about the day-to-day tasks involved in managing and monitoring the cluster. The chapter concludes by showing how to add support for new applications for which there is no third-party agent from Oracle or the application vendor.

Installing the Oracle Solaris OS on a Cluster Node

You can install the Oracle Solaris OS in these ways:

- Interactively using the `installer` program
- With Solaris JumpStart to automate the process
- With public domain or third-party automation tools such as JumpStart Enterprise Toolkit [JetWiki]

If you choose an automated option, then you can perform the installation using collections of Solaris packages or a Solaris Flash archive file that is created using the `flarcreate` command from a previous template installation (see the `flarcreate`(1M) man page) [FlarInstall].

Note

Although it is technically feasible to install a cluster node from a Flash archive that already includes preconfigured Solaris Cluster software, Oracle does not support doing so. You can, however, use an archive with the unconfigured Solaris Cluster packages added.

If your root disk is not already protected by some form of hardware RAID, you should mirror it to protect your node against disk failure. You can protect your root disk using the bundled Solaris Volume Manager software or with the Oracle Solaris ZFS file system. ZFS is available only when you use the Solaris 10 OS. If you have a software license for Veritas Volume Manager, you can also use that to protect your root disk.

Solaris Live Upgrade eases the process of upgrading your systems. Solaris Live Upgrade is available in all Solaris releases beginning with the Solaris 8 OS. However, the combination of the Solaris 10 OS, ZFS, and Solaris Live Upgrade provides the simplest and most flexible upgrade option for your systems.

When choosing hostnames for your cluster nodes, you must ensure that they comply with RFC 1123 [RFC1123]. If you intend to install Solaris Cluster Geographic Edition, you must not use an "_" (underscore) in the hostnames. Other applications might place additional constraints on the hostnames. Changing a hostname after a cluster has been installed is a complex process and should be avoided, if possible.

Although these sections on installing the Solaris Cluster software cover many important points, they do not provide all the steps you need to follow. Therefore, you must consult the latest versions of the Oracle Solaris OS [S10InstallGuide] and the Solaris Cluster software [SCInstallGuide] installation guides.

Root Disk Partition Requirement for the Solaris Cluster Software

You can use either UFS or ZFS as the root (/) file system for a Solaris Cluster node. The way you partition your root disks depends on which file system you choose and whether you also use Solaris Volume Manager.

After installation of the Solaris Cluster software, each node has a 512-megabyte `/global/.devices/node@`*X* file system that is mounted globally (the *X* represents the node number). To achieve this configuration, you can allow the `scinstall` program to create a `lofi`-based file system for you that works for both UFS and ZFS root (/) file systems. Alternatively, if you use UFS for the root (/) file system, you can create and mount a 512-megabyte file system on `/globaldevices` and allow the `scinstall` program to reuse it for the `/global/.devices/node@`*X* file system.

If you intend to use Solaris Volume Manager, then you must create the main state replica databases. These databases also require separate disk partitions, which should be 32 megabytes in size. This partition is usually placed on slice 7 of the root disk. This can pose a problem if you are using ZFS for the root disk because the standard Solaris installation uses the entire disk for the root zpool, unless you pre-partition your disk before the installation process begins. You can achieve a root disk layout where slice 7 is 32 megabytes in size, and use a ZFS root (/) file system, if you install your system using JumpStart Enterprise Toolkit (JET). However, you must not use ZFS volumes (zvols) to store the Solaris Volume Manager state replica databases, as this configuration is not supported.

Because Solaris Volume Manager relies on a state replica majority (see the section "Solaris Volume Manager's State Replica Majority" in Chapter 1, "Oracle Solaris Cluster: Overview") to maintain data integrity if a disk failure occurs, you should assign slices from three separate, nonshared disks. If you do not have three separate slices available, you can place all of the replicas on a single disk as shown in the following example.

Example 4.1 Creating the Solaris Volume Manager Root State Replica Databases

Use the `metadb` command to create the Solaris Volume Manager root state replica databases.

```
# metadb -afc 3 c1t0d0s7
# metadb
        flags           first blk       block count
     a m  pc luo        16              8192            /dev/dsk/c1t0d0s7
     a    pc luo        8208            8192            /dev/dsk/c1t0d0s7
     a    pc luo        16400           8192            /dev/dsk/c1t0d0s7
```

If you use Veritas Volume Manager, you ideally use it only for shared disk management, and you use either Solaris Volume Manager or the Oracle Solaris ZFS file system to mirror your root disks.

Planning for Upgrades

After you have installed your cluster, you must maintain it through its lifecycle. Doing so inevitably requires that you install both Solaris and cluster patches. You might also perform more major changes such as upgrading to a new Solaris Cluster release. To minimize any disruption, you must plan ahead.

"Patching and Upgrading Your Cluster" describes in more detail the options for ongoing cluster maintenance. If you choose ZFS for your root disk, then the upgrade procedure is fairly straightforward because Solaris Live Upgrade can create an alternate boot environment on the existing root zpool. However, if you choose UFS, then you must have the same number of slices available as your existing root disk has, which usually means using a separate disk. However, if you have only a root partition, you can plan for future upgrades by setting aside a separate partition of the same size on the root disk.

Securing Your Solaris Operating System

The security of your Solaris Cluster system must be foremost in your mind. Without security measures in place, your system could be vulnerable to denial-of-service attacks or have its data stolen, compromised, or deleted. Any of these threats could lead to your service being unavailable. Therefore, you must put barriers in place to prevent such threats.

Although several methods for auditing security violations are available, this section focuses on how to reduce the number of targets that an unauthorized user can attack. Even when your cluster is in a physically secure location, you can still take the additional measures of operating system minimization and hardening.

Note

The Solaris Cluster software does not yet support Solaris 10 Trusted Extensions.

Operating Environment Minimization

With operating system minimization, you install the fewest number of Solaris packages that are required to operate your cluster and the services it supports. The Solaris installation process allows you to choose which software groups you install. These groups are shown in Figure 4.1. Furthermore, you can customize the installation to the level of individual packages.

Clearly, choosing the software group at the center of the diagram means installing the fewest packages and thus presents the smallest "attack profile" to potential intruders. However, this installation is insufficient for the Solaris Cluster 3.2 1/09 software, which requires at least the End User Solaris Software Group (`SUNWCuser`) [SCInstallGuide].

One problem you face when trying to minimize your Oracle Solaris OS is that of ongoing maintenance, particularly with new or changing application require-

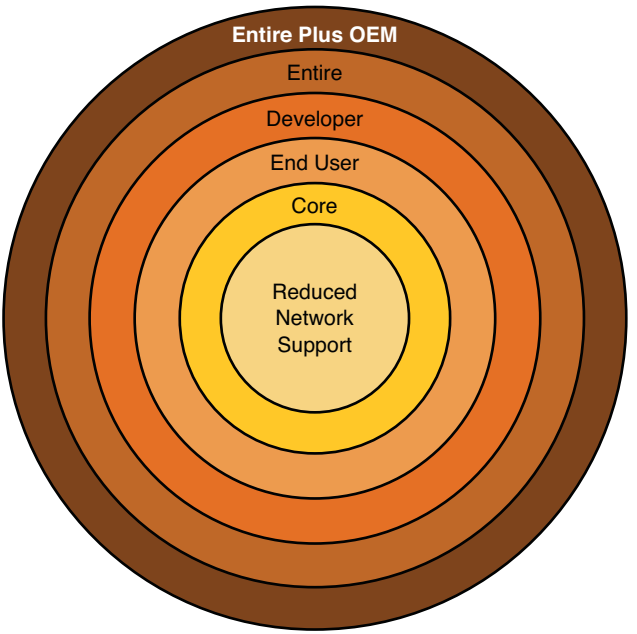

Figure 4.1 Solaris software groups for installation

ments. Though your initial system might well be served by just the `SUNWCuser` package group, you could find that a new service requires some additional packages. Not only does this mean that you need to install the missing packages, but you might also have to reapply certain patches that would have been applied to these packages had they been installed in the first place. This also assumes that the Solaris packages on which your applications are dependent are clearly documented.

For these reasons, you must take a pragmatic approach to minimization. If your systems are unlikely to change over time, and testing has confirmed that the `SUNWCuser` group is sufficient for your needs, then installing that group should pose no problems. Conversely, if the services hosted on your cluster are likely to change, then installing the Entire package group will better serve your needs.

Regardless of your starting point, any subsequent change to the Solaris package installation list must be performed under a strong change management control process. Such a process would require thorough testing of your services in the new environment and verifying that they can still switch over successfully. Because such testing is disruptive, any preparatory work must be performed in a test environment before final testing during a planned maintenance period.

Operating System Hardening

In contrast to operating system minimization, which reduces the number of packages installed on the system, operating system hardening disables or modifies the configuration of services that are installed on the system. An example is the disabling of the `rlogin`, `rsh`, and `rcp` commands in preference to their secure counterparts, `ssh` and `scp`.

As with Oracle Solaris OS minimization, there is clearly an almost unlimited scope for OS hardening. Unfortunately, not all changes are necessarily compatible with running the Solaris Cluster software. Furthermore, if you have made several custom changes to your system, it might prove tricky for Oracle's support services to determine whether any fault that might arise has a root cause in your hardening choices or in the Solaris Cluster software itself. Consequently, they might request that you reverse your hardening effort in order to facilitate their diagnosis. Doing so could be complicated if you do not have a well-documented change control process or an automated system for applying your hardening rules.

Fortunately, Sun has developed a supported system to help you harden your systems effectively. The package is known as the Solaris Security Toolkit (formally known as the JumpStart Architecture and Security Scripts, or JASS) [SSTAdminGuide]. This allows you to harden your Oracle Solaris OS in a supported and reversible manner, regardless of whether you are using the Solaris 9 OS or the Solaris 10 OS.

In addition, in the Solaris 10 OS 11/06 release comes the Secure by Default configuration [SBDBlog]. This configuration does not make the Solaris Security Toolkit obsolete, but it does subsume many of the settings that the toolkit previously changed as part of the hardening process. The latest version of the toolkit still provides fine-grained control of your hardening process and performs many other security-related tasks.

With Secure by Default configuration, you need to relax some of these settings in the Solaris 10 OS prior to installing the Solaris Cluster software. The commands you need to execute are provided in the following example [ThorstenBlog].

Example 4.2 Reversing Some of the Solaris 10 Secure by Default Settings for a Solaris Cluster Installation

Ensure that the `local_only` property of `rpcbind` is set to `false`.

```
# svcprop network/rpc/bind:default | grep local_only
```

If `local_only` is not set to `false`, run

```
# svccfg
svc:> select network/rpc/bind
svc:/network/rpc/bind> setprop config/local_only=false
svc:/network/rpc/bind> quit
# svcadm refresh network/rpc/bind:default
```

This value is needed for cluster communication between nodes.

Ensure that the `tcp_listen` property of `webconsole` is set to `true`.

```
# svcprop /system/webconsole:console | grep tcp_listen

If tcp_listen is not true, run

# svccfg
svc:> select system/webconsole
svc:/system/webconsole> setprop options/tcp_listen=true
svc:/system/webconsole> quit
# svcadm refresh svc:/system/webconsole:console
# /usr/sbin/smcwebserver restart
```

This value is needed for Solaris Cluster Manager communication.

To verify that the port is listening to `*.6789`, run

```
# netstat -an | grep 6789
```

Securing Network Communications

To secure your cluster on the network, start from the position of denying access to everything. Then proceed to allow access to ports that are specifically required by your applications and ports that are required by your management and administration procedures.

To secure network communications, you can use external firewalls. Alternatively, if you are using the Solaris 10 OS, you can use the built-in `ipfilter` capabilities. If you plan to use scalable services with shared IP addresses, you must use an external firewall. This is because the `ipfilter` software (see the `ipfilter`(5) man page) cannot detect the connection state after the shared address migrates from one cluster node to another cluster node.

In addition, if the applications on your cluster transmit sensitive data over the network to client systems, consider using the IP Security Architecture (IPsec) to protect the communication traffic (see the `IPsec`(7P) man page).

Solaris Cluster Software Installation

After you have installed and secured the Oracle Solaris OS, you can install the Solaris Cluster software. If you use Solaris JumpStart, then you can set up the JumpStart

server to install the Solaris Cluster software as part of the postinstallation scripts. This Solaris Cluster software installation process installs only the relevant packages for you and does not actually configure the software. To achieve this level of automation, you can add your own scripts to the JumpStart framework. Many installations performed by Sun Professional Services use the freely available JumpStart Enterprise Toolkit (JET). JET enables you to create individual modules to perform specific tasks such as installing or configuring software. One such module, available as part of certain Sun Professional Services engagements, can install and perform a basic configuration of the Solaris Cluster software.

The installation process has two parts: installing the software and configuring the software. The latter requires your nodes to be rebooted as part of that process. Software installation is performed through the `installer` program found on the installation media.

The `installer` program can be run in both character mode and as a full graphical user interface (GUI). To run the GUI `installer` program, ideally use `ssh` with X forwarding enabled. If you choose not to use `ssh`, you can use `telnet` or `rlogin` in combination with the `xhost` command to allow the cluster client node to display on the X server display you are using. If you use `telnet` or `rlogin`, you must set the `DISPLAY` environment variable to the workstation from which you are accessing the cluster.

When run from the command line, the `installer` program can be supplied with a text-based "response" file to allow automation of the Solaris Cluster software installation.

Example 4.3 Starting the Solaris Cluster Software `installer` Program

Use the `-X` flag to `ssh` to forward the X11 protocol to your local display.

```
admin_console# ssh -X root@phys-earth1
Password:
Last login: Thu Nov 12 08:19:29 2009 from admin_console.
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
```

Check that `ssh` is configured to forward correctly.

```
cluster1# grep X11F /etc/ssh/sshd_config
X11Forwarding yes
```

Change to the Solaris Cluster media directory and run the installer.

```
cluster1# cd DVD-mount-point
cluster1# ls
Copyright      License        README         Solaris_sparc  Solaris_x86
```

```
cluster1# cd Solaris_sparc/
cluster1# ls -l
total 28
drwxr-xr-x   9 root     root           9 Aug  6 16:17 Product
-rwxr-xr-x   1 root     root       10892 Jan  8  2009 installer
-rw-r--r--   1 root     root          84 Jan  8  2009 release_info
cluster1# ./installer
```

Assuming you are installing from the installation media, after you have accepted the software license agreement, a wizard screen similar to Figure 4.2 is displayed.

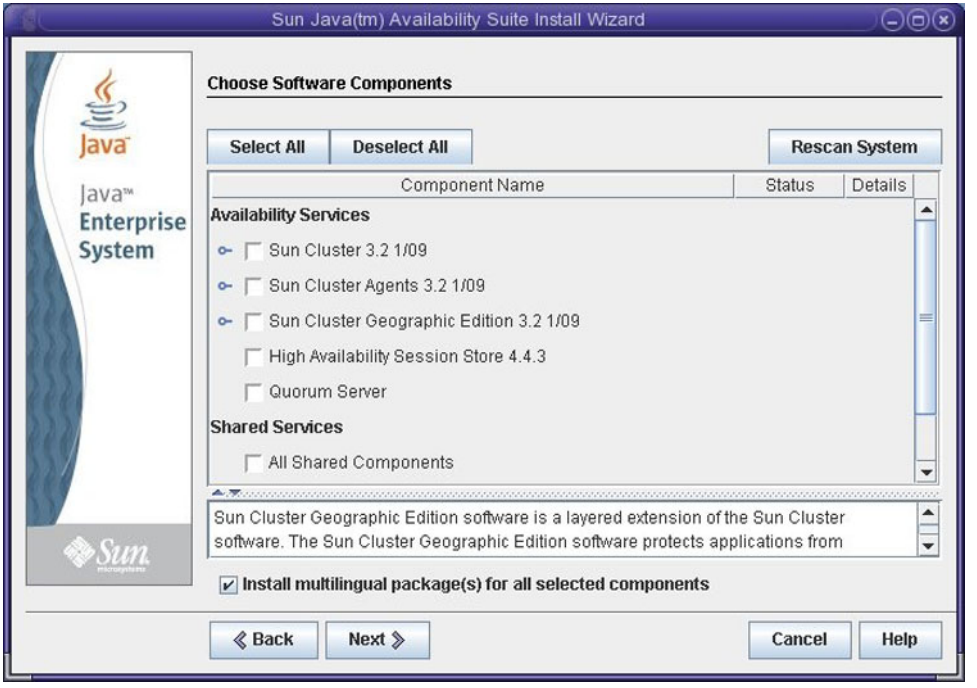

Figure 4.2 Installation screen for the Solaris Cluster 3.2 01/09 software

In the wizard, select the check boxes for the software you have chosen to install. Note that the option to install the multilingual packages is checked by default, whereas with the text `installer` program, you must explicitly choose to install these packages. Follow the wizard instructions to complete the installation. You must perform this installation on all the nodes that will form your cluster.

After the packages have been installed on all the nodes, you can proceed with the configuration of the cluster itself by using the `scinstall` command. You have several configuration options to choose from. The simplest option configures the Solaris Cluster software on just one node. The most automated option automatically configures the Solaris Cluster software on all your cluster nodes from the single command.

As each node is configured, it is rebooted by the configuration process. Therefore, do not run this program unless your nodes can experience an outage at this time.

If the configuration is performed by a certified Solaris Cluster installer, he or she will follow Oracle System Installation standards methodology to complete the installation. Doing so mostly involves checking that certain settings have been made and that the appropriate patches have been installed. If you choose to install the cluster yourself, you must purchase a cluster validation service from Sun Services to ensure that your installation complies with the necessary standards for it to be supportable.

After all the cluster nodes have been correctly configured and validated, you can begin the process of creating the resource groups and resources that encapsulate the services you choose to make highly available.

Time Synchronization

Every aspect of managing, securing, planning, and debugging a network involves determining when events happen. Time is the critical element that enables an event on one network node to be mapped to a corresponding event on another network node. In many cases, these challenges can be overcome by the enterprise deployment of the Network Time Protocol (NTP) service. You can configure this service to operate in the following modes: NTP server, NTP client, or NTP peer.

The `xntpd` daemon (see the `xntpd`(1M) man page) is bundled with the Solaris software to provide time synchronization services to all cluster nodes. The Solaris Cluster software installation creates a `/etc/inet/ntp.conf.cluster` file on each cluster node and a legacy `/etc/rc2.d/S74xntpd_cluster` script (or legacy Solaris Management Facility [SMF] service) to start `xntpd` with this file, instead of the default `/etc/inet/ntp.conf` file. The standard Solaris 10 SMF service, `/network/ntp:default`, is disabled.

The `ntp.conf.cluster` file, created by the Solaris Cluster configuration process, synchronizes the time across all cluster nodes by naming the cluster nodes as peers. You can also synchronize your cluster nodes to an external NTP server either

by naming a specific NTP server or by broadcasting or multicasting for one. To do this, you can specify the appropriate directive in the `/etc/inet/ntp.conf.cluster` file on each cluster node. Because the `S74xntpd_cluster` script calls `ntpdate` (see the `ntpdate`(1M) man page), do not restart this service while the cluster is running. If the cluster nodes are not closely synchronized with the external source, restarting the service will result in a large and sudden change in the system clocks, either backward or forward. Under some circumstances, this change can result in a system panic. Instead, follow the recommended procedure, which involves rebooting the cluster. Clearly, you should schedule such a task for the next maintenance window.

Warning

When using NTP, do not attempt to adjust the cluster time while the cluster is running. Do not adjust the time by interactively using the `date`, `rdate`, `xntpd`, or `svcadm` command or within `cron` scripts.

Cluster Management

For the most part, you can manage your cluster using either the command-line interface (CLI) or the Solaris Cluster Manager GUI. Although the capabilities provided by the GUI are a subset of what is available through the CLI, its graphical nature makes it less prone to user errors when creating or modifying complex objects. This is particularly true if a resource has many mandatory parameters or if a resource group requires several affinities to be configured.

Command-Line Interface

The Solaris Cluster CLI commands are in the `/usr/cluster/bin` directory, and all begin with the `cl` prefix. Most of these commands have both a long form and a short form. The directory also contains the CLI commands used to manage systems installed with software releases prior to the Solaris Cluster 3.2 software, such as `scrgadm`, `scconf`, and `scswitch`.

All of the commands listed in Table 4.1, excluding `clsetup`, which is menu-driven, conform to this format: `command` *`action argument ... operand`*.

All the commands return an exit code of zero on success, allowing you to create scripts to automate procedures that you might perform regularly.

Table 4.1 Solaris Cluster Commands

CLI— Long Form	CLI— Short Form	Description
`claccess`	`claccess`	Manages the Solaris Cluster access policies for nodes
`cldevice`	`cldev`	Manages the Solaris Cluster devices
`cldevicegroup`	`cldg`	Manages the Solaris Cluster device groups
`clinterconnect`	`clintr`	Manages the Solaris Cluster private interconnects
`clnasdevice`	`clnas`	Manages access to NAS devices from the Solaris Cluster software
`clnode`	`clnode`	Manages the Solaris Cluster nodes
`clquorum`	`clq`	Manages the Solaris Cluster quorum devices and properties
`clreslogicalhostname`	`clrslh`	Manages resources for the Solaris Cluster logical hostnames
`clresource`	`clrs`	Manages resources for the Solaris Cluster data services
`clresourcegroup`	`clrg`	Manages resource groups for the Solaris Cluster data services
`clresourcetype`	`clrt`	Manages resource types for the Solaris Cluster data services
`clressharedaddress`	`clrssa`	Manages the Solaris Cluster resources for shared addresses
`clsetup`	`clsetup`	Menu-driven command used to configure the Solaris Cluster software interactively
`clsnmphost`	`clsnmphost`	Administers the list of Solaris Cluster Simple Network Management Protocol (SNMP) hosts
`clsnmpmib`	`clsnmpmib`	Administers the Solaris Cluster SNMP management information bases (MIBs)
`clsnmpuser`	`clsnmpuser`	Administers the Solaris Cluster SNMP users
`cltelemetryattribute`	`clta`	Configures system resource monitoring
`cluster`	`cluster`	Manages the global configuration and status of a cluster
`clvxvm`	`clvxvm`	Configures VxVM for the Solaris Cluster software
`clzonecluster`	`clzc`	Creates and manages zone clusters

The Solaris Cluster Manager Graphical User Interface

You can run the Solaris Cluster Manager GUI using a browser such as Mozilla Firefox or Microsoft Internet Explorer. Solaris Cluster Manager requires that you have already installed

- The common agent container (CAC) package (`SUNWcacaort`)
- The Solaris Cluster Manager packages (`SUNWscspmr`, `SUNWscspmu`)
- The Sun Java Web Console packages (`SUNWmcon`, `SUNWmconr`, `SUNWmctag`)

After the Sun Java Web Console is started, it listens for secure `https` connections on port 6789, if you have reversed the hardening performed by the standard Solaris installation process (see Example 4.2). In addition, at least version 2.2 of the CAC service must be running. CAC listens for Java Management Extensions (JMX) calls made by the Sun Java Web Console as a result of actions you perform through the Solaris Cluster Manager GUI.

Example 4.4 Checking That the Required Components Are in Place to Run Solaris Cluster Manager

Check that the required packages are present.

```
# pkginfo SUNWcacaort
application SUNWcacaort Common Agent Container - Runtime

# pkginfo | grep SUNWscspm
application SUNWscspmr                      Sun Cluster Manager (root)
application SUNWscspmu                      Sun Cluster Manager (Usr)

# pkginfo SUNWmcon SUNWmconr SUNWmctag
application SUNWmcon  Sun Java(TM) Web Console 3.0.2 (Core)
system      SUNWmconr Sun Java(TM) Web Console 3.0.2 (Root)
application SUNWmctag Sun Java(TM) Web Console 3.0.2 (Tags & Components)
```

Check the CAC service version.

```
# cacaoadm -V
2.2.0.1
```

Check that CAC is online.

```
# svcs /application/management/common-agent-container-1
STATE          STIME    FMRI
online          7:58:28 svc:/application/management/common-agent-container-1:default
```

Check that the Java Web Console is online.

```
# svcs svc:/system/webconsole:console
STATE          STIME    FMRI
online         Jan_19   svc:/system/webconsole:console
```

Check that the Java Web Console is listening on all networks to port 6789.

```
# netstat -a | grep 6789
      *.6789               *.*                0      0 49152      0 LISTEN
```

Check the applications deployed to the Java Web Console.

```
# wcadmin list

Deployed web applications (application name, context name, status):

    SunClusterManager  SunClusterManager  [running]
    console            ROOT               [running]
    console            com_sun_web_ui     [running]
    console            console            [running]
    console            manager            [running]
    legacy             SunPlexManagerGeo  [running]
    zfs                zfs                [running]

Registered jar files (application name, identifier, path):

    SunClusterManager cacao_jars    /usr/lib/cacao/lib/*.jar
    SunClusterManager jdmk_jars     /opt/SUNWjdmk/5.1/lib/*.jar
    console           audit_jar     /usr/lib/audit/Audit.jar
    console           console_jars  /usr/share/webconsole/lib/*.jar
    console           jato_jar      /usr/share/lib/jato/jato.jar
    console           javahelp_jar  /usr/jdk/packages/javax.help-2.0/lib/*.jar
    console           shared_jars   /usr/share/webconsole/private/container/shared/
lib/*.jar

Registered login modules (application name, service name, identifier):

    console  ConsoleLogin  userlogin
    console  ConsoleLogin  rolelogin

Persistent Jvm options:

    -XX:ParallelGCThreads=4
    -server
    -Xmx256m
    -XX:+BackgroundCompilation
    -XX:+UseParallelGC

Shared service properties (name, value):

    ENABLE            yes
    java.options      -server -XX:+BackgroundCompilation -Xmx256m
```

After you have logged in to the Java Web Console and chosen the Solaris Cluster Manager option, a screen similar to Figure 4.3 is displayed. Using Solaris Cluster Manager, you can perform a wide range of tasks, including

- Creating, modifying, and deleting resource groups and resources
- Enabling and disabling resources
- Bringing resource groups online and offline
- Switching resource groups between nodes

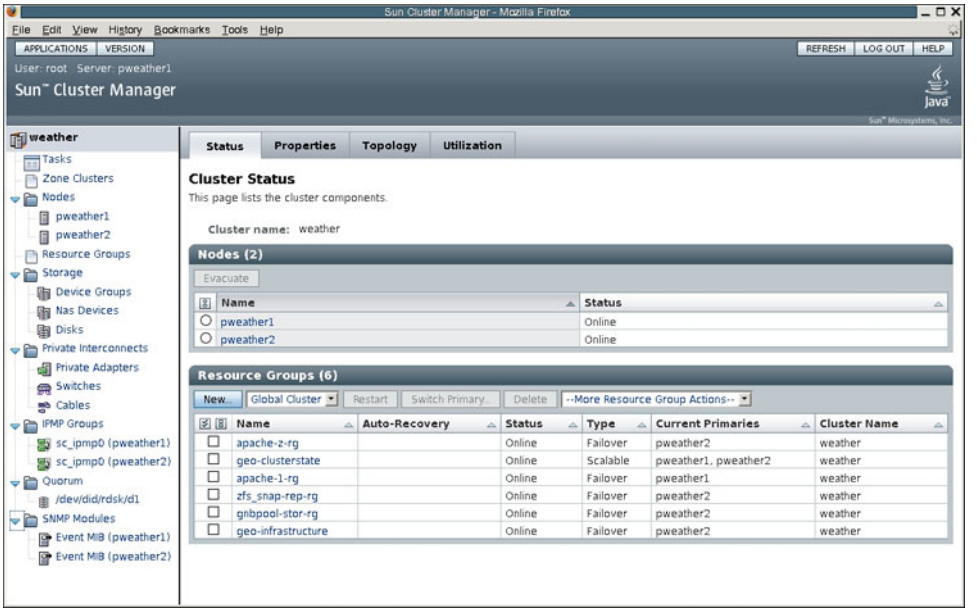

Figure 4.3 Solaris Cluster Manager browser-based GUI

Solaris Cluster Manager also highlights faults in the system with a red marker over the components that have errors.

Solaris Cluster Wizards

As with the `clsetup` menu-driven command, Solaris Cluster Manager has several wizards to help you set up some common highly available services, such as Network File System (NFS), Apache web server, and Oracle databases, as well as two more complex services: Oracle Real Application Clusters (Oracle RAC) and SAP web application server.

The wizards simplify your work by probing the system to discover the potential options that are available to you and then configuring the relevant resource groups

and resources with the appropriate dependencies and affinities. A good example of this process is the Oracle RAC wizard. It can do the following:

- Create the RAC framework resource group and resources for you, depending on your answer regarding your volume management method (hardware RAID, Solaris Volume Manager for Sun Cluster, or the VxVM cluster feature), shared QFS file system, or another supported combination.
- Create the necessary scalable storage resource groups and resources.
- Create the RAC server proxy resource group and resource based on the Oracle RAC version, the `ORACLE_HOME` variable, and the `ORACLE_SID` variables you select. Again, most of these choices are discovered for you by the wizard.
- Create the Oracle Clusterware resources that integrate with the Solaris Cluster storage resources to give the Oracle RAC database instances visibility into the availability of their underlying storage.

The wizard completes the operations for you and displays the Solaris Cluster commands it used to achieve these steps. If anything goes wrong during the creation of these objects, then the wizard will undo the changes.

Role-Based Access Control

By default, only the Solaris root user can manage the cluster. Non-root users cannot create, modify, or delete resource groups or resources, nor can they switch resource groups between the nodes. Non-root users can view the resource group and resource configuration because they are assigned the `solaris.cluster.read` rights profile as part of their standard authorizations. To delegate some of the management functions to specific system administrators who are not assigned full root access, you can use the role-based access control (RBAC) capabilities to assign them the necessary `solaris.cluster.admin` or `solaris.cluster.modify` Solaris Cluster management rights profile.

The following example shows you how to assign to the Solaris user `myadmin` all the Solaris Cluster management rights profiles, which include both `solaris.cluster.admin` and `solaris.cluster.modify`. Initially, without being assigned the `solaris.cluster.modify` rights profile, the user could not create the resource group `foo-rg`. After the user is assigned this rights profile, the user can create and delete the resource group. However, in order to manage or bring the resource group online, the user also needs to have the `solaris.cluster.admin` rights profile assigned. There is no simple way to assign roles such that individual users can control individual resource groups or resources.

Example 4.5 Assigning a Solaris Cluster Management Rights Profile to a User

Use the `id` and `auths` commands to determine the current user ID and authorizations.

```
# id -a
uid=424242(myadmin) gid=424242(myadmin) groups=424242(myadmin)
# clresourcegroup list
apache-1-rg
geo-clusterstate
geo-infrastructure
apacheset-stor-rg
seasons_apache-rep-rg
apache-z-rg
# auths
      .
      .
      .
...solaris.cluster.read,solaris.cluster.device.read,solaris.cluster.network.read,
solaris.cluster.node.read,solaris.cluster.quorum.read,solaris.cluster.resource.read,
solaris.cluster.system.read,solaris.cluster.transport.read,solaris.cluster.gui,solaris
.cluster.geo.read
# clresourcegroup create foo-rg
clrg:  (C368238) You are not authorized to run this command with these options.
clrg:  (C666174) An RBAC authorization of "solaris.cluster.modify" is required.
clrg:  (C546315) Refer to the rbac(5) man page for more information about RBAC
authorizations.
```

Modify the `/etc/user_attr` file to include the required rights.

```
# grep myadmin /etc/user_attr
myadmin::::auths=solaris.cluster.*;
# id -a
uid=424242(myadmin) gid=424242(myadmin) groups=424242(myadmin)
# auths
solaris.cluster.*,
      .
      .
      .
solaris.cluster.read,solaris.cluster.device.read,solaris.cluster.network.read,solaris.
cluster.node.read,solaris.cluster.quorum.read,solaris.cluster.resource.read,solaris.
cluster.system.read,solaris.cluster.transport.read,solaris.cluster.gui,solaris.cluster
.geo.read
# clresourcegroup create foo-rg
# clresourcegroup list
apache-1-rg
geo-clusterstate
geo-infrastructure
apacheset-stor-rg
seasons_apache-rep-rg
apache-z-rg
foo-rg
# clresourcegroup delete foo-rg
```

Just as these rights profiles can be assigned to a particular user, they can also be assigned to roles, which a user can assume. In the following example, the role `clusadm` is created. Unless the user `myadmin` is assigned access to that role, the user is unable to assume it through the `su` command, even though he or she might

have the right password. However, after being assigned the role, the user can perform the actions that the role is authorized to perform.

Example 4.6 Using an RBAC Role to Allow a User to Perform Cluster Management Tasks

Display the `Cluster Management` entry in the profile description database.

```
# grep "Cluster Management" /etc/security/prof_attr | head -1
Cluster Management:::Sun Cluster
Management:auths=solaris.cluster.admin,solaris.cluster.modify,
      .
      .
      .
# roleadd  -P "Cluster Management"  clusadm
# tail -3 /etc/user_attr
root::::auths=solaris.*,solaris.grant;profiles=Web Console Management,All;lock_after_
retries=no;min_label=admin_low;clearance=admin_high
zfssnap::::type=role;auths=solaris.smf.manage.zfs-auto-snapshot;profiles=ZFS File
System Management
clusadm::::type=role;profiles=Cluster Management
# passwd clusadm
New Password:
Re-enter new Password:
passwd: password successfully changed for clusadm
# su - myadmin
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
$ su clusadm
Password:
Roles can only be assumed by authorized users
su: Sorry
$ logout
# usermod -R clusadm myadmin
# tail -3 /etc/user_attr
zfssnap::::type=role;auths=solaris.smf.manage.zfs-auto-snapshot;profiles=ZFS File
System Management
clusadm::::type=role;profiles=Cluster Management
myadmin::::type=normal;roles=clusadm
# su - myadmin
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
$ su clusadm
Password:
$ clresourcegroup create bar-rg
$ clresourcegroup delete bar-rg
```

Cluster Monitoring

Most data centers use an enterprise management tool such as HP OpenView, BMC Patrol, or IBM Tivoli to monitor and manage their environment. If the data center has a large base of Sun servers, then the list might also include the Sun Management Center software. The Solaris Cluster software integrates with these tools using the Simple Network Management Protocol (SNMP), enabling administrators to monitor both the cluster nodes and the resource groups and resources they contain.

Sun Management Center Integration

The Sun Management Center software enables you to manage and monitor all the Sun servers in your data center. The Sun Management Center software is implemented as a three-tier architecture: console, server, and agent. The agent is installed on the system you choose to monitor. It sends alerts to the server when certain events occur or when thresholds are exceeded. Each agent handles the monitoring of a specific function. For example, the Solaris Cluster agent monitors the cluster infrastructure and the resource groups and resources configured on your system. The Sun Management Center console aggregates the alarms and alerts pertaining to a specific node, marking the node with a symbol to denote the most serious condition that it is encountering. If there are no problems with the node, then no symbol is present. Figure 4.4 shows an example of a serious error with the cluster transports, but just a warning for the scalable resource group because it is offline.

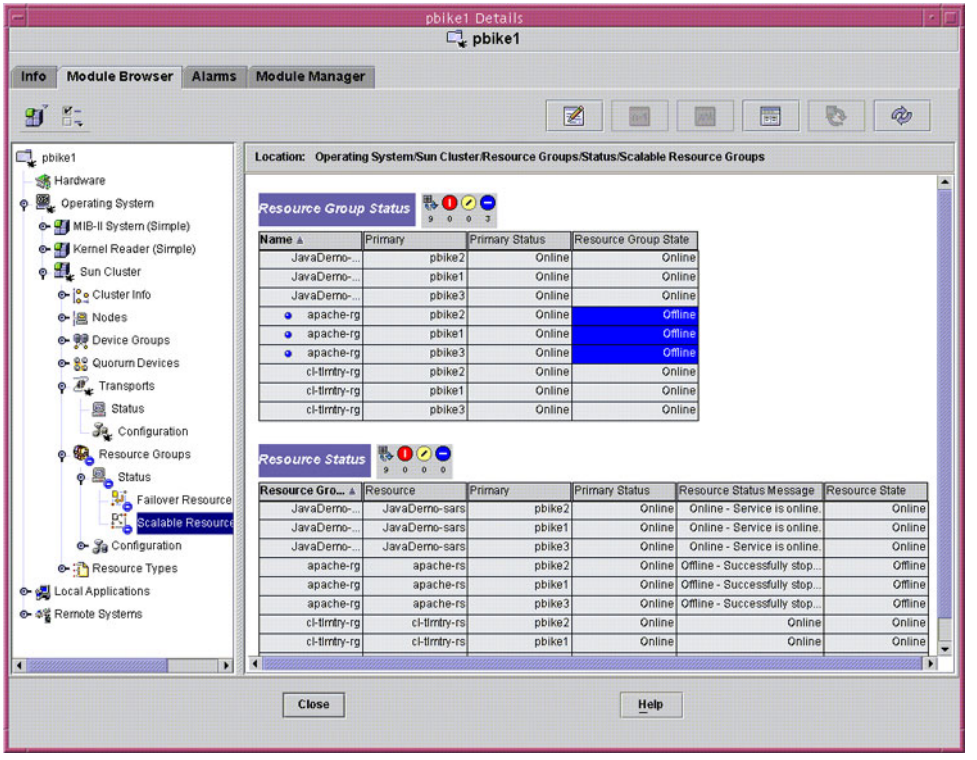

Figure 4.4 Sun Management Center GUI

Solaris Cluster SNMP Management Information Base

The Solaris Cluster software provides two Simple Network Management Protocol (SNMP) management information bases (MIBs): one for cluster events, the other for cluster objects and their status.

SNMP MIB for Cluster Events

The Solaris Cluster SNMP event MIB can be enabled on any cluster node without relying on additional SNMP agent infrastructure. Because all cluster nodes receive the same cluster events, you need to enable the SNMP event MIB on only one cluster node.

The cluster events are graded in severity from low to high, as follows:

- `INFO`
- `WARNING`
- `ERROR`
- `CRITICAL`
- `FATAL`

When the SNMP event MIB is enabled on a node, the cluster event SNMP interface can do the following:

- Store up to 50 of the most recent `WARNING` or higher-severity cluster events in the MIB table.
- Send SNMP traps to one or more hosts when `WARNING` or higher-severity cluster events occur.

By default, the SNMP event module uses port number 11161, and traps use port number 11162. These port numbers can be changed by using the `cacaoadm` command, as shown in the following example.

The Solaris Cluster SNMP event MIB Object ID (OID) is `1.3.6.1.4.1.42.2.80`.

Example 4.7 Changing the SNMP Adapter Trap Port

Use the `cacaoadm` command to determine and change the SNMP adapter trap port.

```
phys-grass2# cacaoadm get-param snmp-adaptor-trap-port
snmp-adaptor-trap-port=11162
phys-grass2# cacaoadm stop
phys-grass2# cacaoadm set-param snmp-adaptor-trap-port=12345
phys-grass2# cacaoadm start
phys-grass2# cacaoadm get-param snmp-adaptor-trap-port
snmp-adaptor-trap-port=12345
```

SNMP Interface for Cluster Objects

The SNMP interface for cluster objects such as nodes, resource groups, and quorum devices, along with their corresponding status, is provided through the Solaris Cluster software to the Sun Management Center integration module. The integration module is installed by default on all cluster nodes as part of the `SUNWscsal` and `SUNWscsam` packages.

To enable the integration module on a cluster node, the Sun Management Center agent infrastructure must be installed and running on that node. After the Sun Management Center agent starts on the node, the integration module can be loaded and enabled through the Sun Management Center console. The Sun Management Center server and console can be installed on one of the cluster nodes, but they are usually installed on another system.

Similar to the SNMP event MIB, the integration module needs to be enabled on only one cluster node to receive cluster SNMP MIB data and traps. The integration module MIB OID is `1.3.6.1.4.1.42.2.80.1.1.1`.

The Sun Management Center agent running on a cluster node sends SNMP traps to the Sun Management Center server whenever the status of an object changes, such as a resource group going online or offline, or becoming managed or unmanaged. The SNMP traps also can be sent directly from the Sun Management Center agent to any other hosts by adding the secondary trap destination through the Sun Management Center `es-trapdest` command. The following example shows how to add `myHost` to the hosts that receive Sun Management Center SNMP traps from the agent on port 11162.

Example 4.8 Adding a Secondary Trap Destination to the Sun Management Center Agent

Add `myHost` as a secondary trap destination using the `es-trapdest` command.

```
# /opt/SUNWsymon/sbin/es-trapdest -c agent -a myHost 11162
```

The agent has to be restarted for the setting to take effect.

Service-Level Management and Telemetry

When you consolidate multiple services onto a Solaris Cluster installation, you must ensure that your service levels are met even when several services reside on the same cluster node. The Oracle Solaris OS has many features, such as resource controls and scheduler options, to help you achieve these goals. These resource

allocations can be defined in the projects database stored locally in `/etc/project` or held in the name service maps.

The Solaris Cluster software can bind both resource groups and resources to projects using the `RG_project_name` and `Resource_project_name` properties, respectively. The following example shows how to create a processor pool (containing four CPUs) that uses the fair share scheduler (FSS). The processor pool is then associated with the user project that limits shared memory usage to 8 gigabytes. The FSS can be enabled by using the `dispadmin -d FSS` command.

Example 4.9 Binding a Resource Group to a Project Associated with a Processor Pool

Determine the number of processors the system has using the `psrinfo` command.

Define a four-CPU-processor set called `oracle_pset` in a temporary file, and then use the file as input to the `poolcfg` command.

```
# psrinfo | wc -l
      24
# cat /tmp/create_oracle_pool.txt
create pset oracle_pset ( uint pset.min = 1 ; uint pset.max = 4)
create pool oracle_pool
associate pool oracle_pool ( pset oracle_pset )
modify pool oracle_pool ( string pool.scheduler = "FSS" )

# poolcfg -f /tmp/create_oracle_pool.txt
```

Instantiate the configuration using the `pooladm` command.

```
# pooladm -c
# pooladm

system default
        string  system.comment
        int     system.version 1
        boolean system.bind-default true
        string  system.poold.objectives wt-load

        pool pool_default
                int     pool.sys_id 0
                boolean pool.active true
                boolean pool.default true
                string  pool.scheduler FSS
                int     pool.importance 1
                string  pool.comment
                pset    pset_default

        pool oracle_pool
                int     pool.sys_id 2
                boolean pool.active true
                boolean pool.default false
                string  pool.scheduler FSS
                int     pool.importance 1
                string  pool.comment
                pset    oracle_pset
```

```
        pset oracle_pset
                int     pset.sys_id 1
                boolean pset.default false
                uint    pset.min 1
                uint    pset.max 4
                string  pset.units population
                uint    pset.load 17
                uint    pset.size 4
                string  pset.comment

                cpu
                        int     cpu.sys_id 1
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 0
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 3
                        string  cpu.comment
                        string  cpu.status on-line

                cpu
                        int     cpu.sys_id 2
                        string  cpu.comment
                        string  cpu.status on-line

        pset pset_default
                int     pset.sys_id -1
                boolean pset.default true
      .
      .
      .
```

Use the `projadd` command to make `oracle_pool` the project pool for user `oracle`.

```
# projadd -p 4242 -K "project.max-shm-memory=(privileged,8GB,deny)" \
>      -K project.pool=oracle_pool user.oracle
# su - oracle
Sun Microsystems Inc.   SunOS 5.10      Generic January 2005
$ id -p
uid=424242(oracle) gid=424242(oinstall) projid=4242(user.oracle)
$ exit
# clresourcegroup create -p RG_project_name=user.oracle oracle-rg
```

Similarly, using the `clzonecluster` command (see the `clzonecluster`(1M) man page), you can bind zone clusters to pools, dedicate or limit the number of CPUs allocated to them, and limit the physical, swap, or locked memory they can use.

Gathering Telemetry from the Solaris Cluster Software

The Solaris Cluster service-level management feature enables you to configure the Solaris Cluster software to gather telemetry data from your cluster. Using this

feature, you can collect statistics on CPU, memory, swap, and network utilization of the cluster node as well as on resource groups and system components such as disks and network adapters. By monitoring system resource usage through the Solaris Cluster software, you can collect data that reflects how a service using specific system resources is performing. You can also discover resource bottlenecks, overloads, and even underutilized hardware resources. Based on this data, you can assign applications to nodes that have the necessary resources and choose which node each application should fail over to.

This feature must be set up using the `clsetup` command. The telemetry data is stored in its own Java DB database held on a failover or global file system that you must provide for its use. After the setup is complete, you can enable the telemetry on the resource groups, choose the attributes to monitor, and set thresholds. Figure 4.5 and Figure 4.6 show the type of output you can receive from using this feature.

Figure 4.5 shows that an alarm has been generated because disk `d4` has exceeded the threshold set for it.

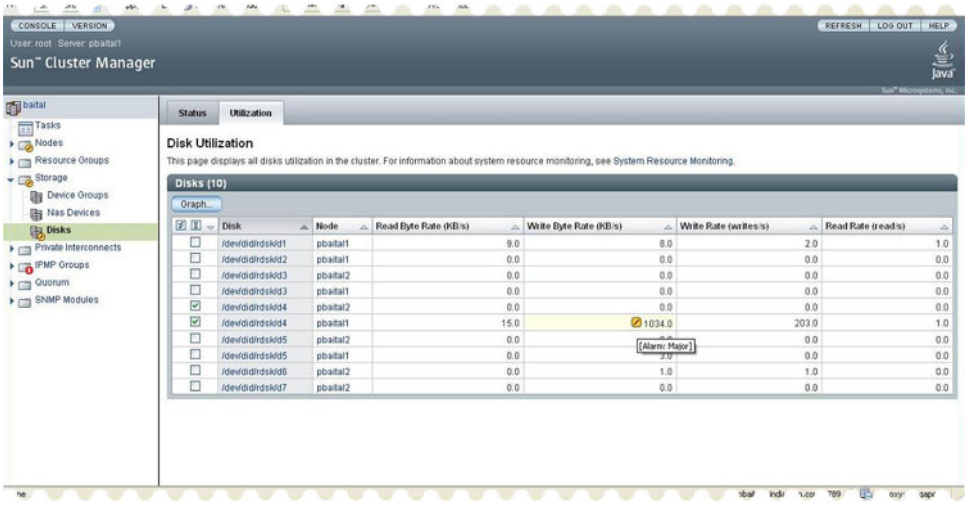

Figure 4.5 Alarm showing that the write I/O rate to disk `d4` has exceeded the threshold set

Figure 4.6 shows the utilization of the public network adapters `bge0` and `bge1` on cluster node `pbaital1`.

The telemetry uses the `RG_slm_type` resource group property, which can be set to one of two values: `automated` or `manual`. The default value for the `RG_slm_type` property is `manual`. Unless the `RG_slm_type` property value is explicitly set to

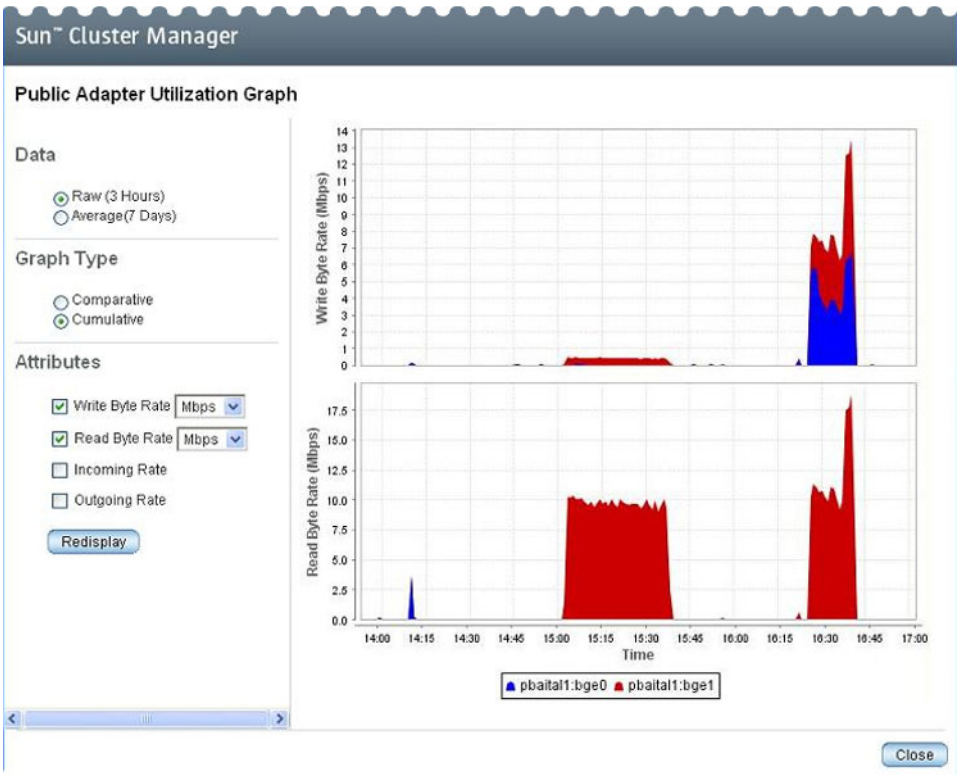

Figure 4.6 Public network adapter utilization telemetry gathered using the service-level management feature

`automated` when a resource group is created, telemetry is not enabled for the resource group. If the resource group `RG_slm_type` property is changed, resource utilization monitoring begins only after the resource group is restarted.

When a resource group has the `RG_slm_type` property set to `automated`, the Resource Group Manager (RGM) internally generates a Solaris project to track the system resource utilization for all processes encapsulated by the resource of the resource group. This tracking happens regardless of whether the `RG_project_name` and `Resource_project_name` properties are set. The telemetry can track only the system resource utilization: CPU usage, resident set size (RSS), and swap usage for resource groups that have the `RG_slm_type` property set to `automated`. Telemetry for other objects is gathered at the node, zone, disk, or network interface level, as appropriate.

See Example 8.9 in Chapter 8, "Example Oracle Solaris Cluster Implementations," for more information about how to set up, configure, and use the Solaris Cluster telemetry.

Using the Solaris Cluster Manager browser interface simplifies the process of configuring thresholds and viewing the telemetry monitoring data.

The following example shows the generated project name in the `RG_SLM_projectname` property. However, unlike other resource group properties, you cannot set this property manually. Furthermore, if `RG_slm_type` is set to `automated`, the `RG_project_name` and `Resource_project_name` properties will be ignored. Conversely, when `RG_slm_type` is set to `manual`, the processes of the resource group's resource will be bound to the projects named in the `RG_project_name` and `Resource_project_name` properties. However, the RGM will not track the system resources they use.

Example 4.10 The Effect of Setting the `RG_slm_type` Property to `automated`

Use the `clresourcegroup` command to show the property settings for the `apache-1-rg` resource group.

```
# clresourcegroup show -v apache-1-rg

=== Resource Groups and Resources ===

Resource Group:                                 apache-1-rg
  RG_description:                                  <NULL>
  RG_mode:                                         Failover
  RG_state:                                        Managed
  RG_project_name:                                 default
  RG_affinities:                                   <NULL>
  RG_SLM_type:                                     manual
  Auto_start_on_new_cluster:                       False
  Failback:                                        False
  Nodelist:                                        phys-winter1 phys-winter2
  Maximum_primaries:                               1
  Desired_primaries:                               1
  RG_dependencies:                                 <NULL>
  Implicit_network_dependencies:                   True
  Global_resources_used:                           <All>
  Pingpong_interval:                               3600
  Pathprefix:                                      <NULL>
  RG_System:                                       False
  Suspend_automatic_recovery:                      False

  --- Resources for Group apache-1-rg ---
      .
      .
      .
```

Use the `clresourcegroup` command to set the `RG_SLM_type` property to `automated`.

```
# clresourcegroup set -p RG_SLM_type=automated apache-1-rg
# clresourcegroup show -v apache-1-rg

=== Resource Groups and Resources ===

Resource Group:                                 apache-1-rg
  RG_description:                                  <NULL>
  RG_mode:                                         Failover
  RG_state:                                        Managed
  RG_project_name:                                 default
  RG_affinities:                                   <NULL>
  RG_SLM_type:                                     automated
  RG_SLM_projectname:                              SCSLM_apache_1_rg
  RG_SLM_pset_type:                                default
  RG_SLM_CPU_SHARES:                               1
  RG_SLM_PSET_MIN:                                 0
  Auto_start_on_new_cluster:                       False
  Failback:                                        False
  Nodelist:                                        phys-winter1 phys-winter2
  Maximum_primaries:                               1
  Desired_primaries:                               1
  RG_dependencies:                                 <NULL>
  Implicit_network_dependencies:                   True
  Global_resources_used:                           <All>
  Pingpong_interval:                               3600
  Pathprefix:                                      <NULL>
  RG_System:                                       False
  Suspend_automatic_recovery:                      False

  --- Resources for Group apache-1-rg ---
      .
      .
      .
```

Patching and Upgrading Your Cluster

The approach you take to patching and upgrading your cluster will vary depending on your data center standards. Some data centers apply only the bare minimum of changes, such as critical security patches, operating on the principle of "if it ain't broke, don't fix it." Clearly, if a system is stable, this is a reasonable approach, especially if availability is your highest priority. Other data centers schedule regular planned outages to perform patching, often choosing to lag a little behind the most recent patches to avoid encountering bad patches. Still others take the "latest and greatest" approach, given that the patches are meant to fix all the key, known issues. Whichever strategy you use, minimizing the overall outage will clearly be your main priority.

You can also upgrade the Solaris Cluster software by patching it with the patches that are built into a later release of the Solaris Cluster software. However, you must follow all the instructions in the README file contained within each patch.

Upgrade Methods

Three main methods are available for patching or upgrading your operating system and Solaris Cluster software: standard upgrade, rolling upgrade, and dual-partition upgrade. Solaris Live Upgrade provides yet more options when used with these methods. Each method results in differing lengths of outage on your cluster, which further depends on the number and type of zones you have configured (see the section "Oracle Solaris Zones" in Chapter 3, "Combining Virtualization Technologies with Oracle Solaris Cluster Software"). The following sections describe these methods.

Before you begin making changes to the system, consider the following checklist:

- Has the upgrade procedure been tested on a test cluster before being implemented on the production system? If not, is management aware of the potential risk of untested procedures?
- Have you verified that the new configuration is supported by all your services? Some applications place greater restrictions on the operating system or other software versions they support.
- Have the proposed changes been agreed upon and recorded in your change control system?
- Do you have a full backup of the cluster prior to making any changes?
- Have you tested that you can switch over all services between cluster nodes prior to your changes? This testing isolates any post-upgrade problems to the upgrade and patches.
- Is there an agreed outage period of a set length?
- Is there a procedure to reverse any changes if the upgrade does not go according to plan? Have you determined when you will need to start backing out any changes if that happens?
- Have you tested that you can switch over all services between cluster nodes after the changes have been made? This testing ensures that if a failure occurs later, your system will continue to provide a set of highly available services.

Standard Upgrade

With standard upgrade, you schedule a complete cluster outage to perform the patch or upgrade process. This method might be considered for applying patches, such as kernel patches, that need to be installed in single-user mode. Major operating system upgrades also fall into this category. You cannot run a Solaris Cluster configuration with nodes running different major versions of the Oracle Solaris OS, for example, Solaris 9 OS and Solaris 10 OS.

A standard upgrade is by far the simplest method but necessitates the longest cluster outage. During this period, your services are unavailable. Consequently, this method is the least desirable from an availability standpoint.

A standard upgrade can be combined with Solaris Live Upgrade. This approach significantly reduces the cluster outage by no longer requiring the entire cluster to be shut down while you are upgrading the software. You first create an alternative boot environment with the upgraded software on each machine (see "Solaris Live Upgrade" for details). When you are ready to activate the new software, halt the entire cluster, and then boot the entire cluster using the alternative boot environment with the new software on each machine.

Rolling Upgrade

With rolling upgrade, the nodes in your cluster are upgraded one at a time. Any service hosted on a node to be upgraded is manually switched over to another cluster node prior to the target node being shut down for patching. After the target node has been patched, it is brought back into the cluster, and the process is repeated until all nodes have been upgraded. After all of the nodes have been upgraded, you execute the `scversions -c` command to commit the cluster to use the new software. The cluster software does not enable cluster-wide protocols until the commit happens. Prior to the commit, all nodes, including nodes with new software, continue to use the old cluster-wide protocols. There is no rolling downgrade after the commit happens. The outage incurred by a service is therefore just twice the time it takes to switch over from one cluster node to another. A switchover of a simple application can take about 30 seconds, though the time varies significantly depending on the application.

A rolling upgrade changes only the Solaris Cluster software or the Oracle Solaris OS. Rolling upgrades can be used as long as the major cluster release and major operating system remain the same, for example, the Solaris Cluster 3.2 software or the Solaris 10 OS. You cannot perform rolling upgrades between major cluster or major OS releases, for example, the Sun Cluster 3.1 software to the Solaris Cluster 3.2 software or the Solaris 9 OS to the Solaris 10 OS.

The rolling upgrade process updates the software on the local storage of a node when that node is not in cluster mode. The highly available (HA) container places the software on shared storage. Because the shared storage is never out of service, the rolling upgrade process has no opportunity to change the software residing on shared storage. Therefore, a rolling upgrade cannot be used to upgrade software on HA containers.

A rolling upgrade can also be combined with Solaris Live Upgrade. This combination does not reduce the service outage. A rolling upgrade without Solaris Live Upgrade reduces the service outage by performing software upgrades while the

machines are not in the cluster. During this time period, the ability of the cluster to survive machine failures is reduced. The use of Solaris Live Upgrade can significantly reduce the time period of increased vulnerability. Solaris Live Upgrade also reduces the period of time in which nodes in the cluster have different software versions. You first create an alternative boot environment with the upgraded software on each machine (see "Solaris Live Upgrade"). You then serially halt each machine and boot that machine from the alternative boot environment with the upgraded software. After you have booted all nodes from the alternative boot environment with the upgraded software, you execute the `scversions -c` command to commit the cluster to use the new software.

Dual-Partition Upgrade

A dual-partition upgrade offers the greatest flexibility in terms of what changes you can make to the system. The change can be a patch, an update release, or a major release. Changes in multiple areas can be applied simultaneously. A dual-partition upgrade can even be used when all the changes are in areas other than the Solaris Cluster software, including the following:

- The Oracle Solaris OS
- A Solaris Cluster software release
- Third-party volume managers and file systems
- Application software (as long as the applications are not installed on shared storage)

A dual-partition upgrade divides your cluster into two parts, each of which is updated separately and sequentially. To invoke a dual-partition upgrade, you must use the `scinstall` command from the target media of the release to which you are upgrading. This command then prompts you with the options shown in Example 4.11. Choose option 3 for a dual-partition upgrade. You must divide the cluster machines into two sets of nodes, which are called *partitions*. You must partition the cluster such that each partition has for each storage device at least one node that can access that storage device. During the upgrade process, each partition will host the cluster applications, and the cluster applications must be able to access storage. If you are unsure of how to partition the cluster, you can choose an option whereby the system presents you with the various ways to group your cluster nodes for the purposes of the upgrade. After you have chosen your partitioning, you next select the option to initiate the actual upgrade.

Example 4.11 Main Menu of the `scinstall` Command Located on the Solaris Cluster Release Media

The main menu of the `scinstall` command provides you with the following options:

```
*** Main Menu ***

    Please select from one of the following (*) options:

        1) Create a new cluster or add a cluster node
        2) Configure a cluster to be JumpStarted from this install server
      * 3) Manage a dual-partition upgrade
      * 4) Upgrade this cluster node
      * 5) Print release information for this cluster node

      * ?) Help with menu options
      * q) Quit

    Option:  3
```

The system software makes changes to the quorum votes, migrates your services to the partition that is still running while the first partition is upgraded, and halts the nodes of the first partition. You boot the nodes of the first partition into non-cluster mode and upgrade the software. After you have completed the upgrade on the first partition, you choose to continue the dual-partition upgrade. The system software reboots the nodes of the first partition into cluster mode as a new cluster. Unlike what happens in an ordinary boot, the nodes of the first partition proceed through the boot process and stop just prior to the point where the cluster would activate resources used by applications (such as mounting file systems and plumbing IP addresses) and start applications. At this point, most of the boot process has completed. The system software halts the nodes of the second partition and waits for them to actually halt. Next, the system software imports volumes, mounts file systems, plumbs IP addresses, and starts applications. In effect, the system performs a switchover of all applications and their resources from the second partition to the new partition. Next, reboot each node of the second partition and upgrade the software. When done with that step, you choose to continue the upgrade. The system software reboots the nodes of the second partition into cluster mode. After all nodes are back in the cluster, the system software restores the quorum configuration and completes the dual-partition upgrade.

If the dual-partition upgrade fails, you boot all nodes in non-cluster mode and execute the `scinstall -u recover` command on each node to restore the initial copy of the Cluster Configuration Repository (CCR). The system is already completely shut down at this point, so a minimal downtime upgrade is not possible. You do not restart the dual-partition upgrade. Instead, you perform a standard upgrade.

The service outage when you perform a dual-partition upgrade on a system with failover application is approximately the same as for a rolling upgrade. Because a time comes when neither partition can host an instance of a scalable application, there is a service outage for the scalable application, and the service outage is approximately the same as that of a failover application. For the simplest scalable applications, the service outage can be about 30 seconds, though the time varies significantly based on the application.

You cannot upgrade any software or data on the shared storage because the services remain running throughout the upgrade process, except for the point when the services are switched over from the partition running the old software to the partition running the new software. Do not make any configuration changes during the dual-partition upgrade. Make configuration changes either before or after the dual-partition upgrade.

A dual-partition upgrade can also be combined with Solaris Live Upgrade. This combination does not reduce the service outage. A dual-partition upgrade without Solaris Live Upgrade reduces the service outage by performing software upgrades while the machines are not in the cluster. During this time period the ability of the cluster to survive machine failures is reduced. The use of Solaris Live Upgrade can significantly reduce the time period of increased vulnerability. You first create an alternative boot environment with the upgraded software on each machine (see "Solaris Live Upgrade"). You then initiate the actual dual-partition upgrade. The system software automatically manages all of the work of the dual-partition upgrade. There no longer is any need to boot into non-cluster mode, so the number of reboots is reduced by half. The net result is that the time required for upgrades is dramatically reduced.

Choosing Which Upgrade Method to Use

The choice of an upgrade method is governed by three factors: the simplicity of the upgrade process, the compatibility of the current and target software environments, and the level of availability your services must achieve during the upgrade process. Thus, if you require simplicity above all else, then the standard upgrade is the best method. If your current and target software releases do not span a major release, and you want to maintain service availability even for scalable services, then you can use a rolling upgrade. Finally, if changes to your cluster environment involve software spanning major releases and you need to minimize your service outage, then you must choose a dual-partition upgrade.

Solaris Live Upgrade

Solaris Live Upgrade is not a stand-alone technology for upgrading a Solaris Cluster system. Solaris Live Upgrade must be used as part of either a standard upgrade, a rolling upgrade, or a dual-partition upgrade. Solaris Live Upgrade

makes it possible to create an alternate boot environment with the upgraded software while the machine continues to run unaffected using the current boot environment. See "Standard Upgrade," "Rolling Upgrade," or "Dual-Partition Upgrade" for details on when to bring the alternate boot environment with the updated software into operational status. The boot environment with the old software is not damaged when you boot the alternate boot environment with the new software. If you encounter a problem with the new software, you can recover by halting all nodes of the cluster and then booting the entire cluster from the boot environment with the old software and configuration information. Solaris Live Upgrade is available for all the operating system releases supported by the Solaris Cluster 3.2 software.

If you are using the Solaris 10 OS with a ZFS root (/) file system, then the effort involved in creating an alternate boot environment is virtually zero because the required new file systems can be created from the existing root zpool. However, if you are using a Solaris Volume Manager or a Veritas Volume Manager mirrored root disk, then the process is significantly more complicated because you must find additional partitions to match the existing root partition allocation. Consequently, using the Solaris 10 OS and ZFS for the root disk has significant advantages. However, do not upgrade the software version of the zpools in your cluster until you are satisfied that your new environment is working correctly. If you do upgrade the zpool version, you will not be able to fail back to your old boot environment, as the old environment might not be able to mount the zpools that already have a higher version number. Such zpools cannot be downgraded.

After you have updated the Oracle Solaris OS on your new boot environment, you can proceed with updating the Solaris Cluster software. When using Solaris Live Upgrade, you must follow the documentation on how to handle specific steps relating to the volume managers, Solaris Volume Manager and Veritas Volume Manager, and which version of the Java Runtime Environment (JRE) is required.

The process for updating the Solaris Cluster software on the alternate boot environment is outlined in the following steps:

1. Create an `installer` program state file to drive the upgrade of the shared components on the alternate boot environment, but without actually performing the actions.
2. Run the `installer` program again using this state file but targeted on the alternate boot environment.
3. Update the Solaris Cluster framework by running the `scinstall` command from the new Solaris Cluster media, again targeting the alternate boot environment.
4. Update the Solaris Cluster agents by running the `scinstall` command from the new Solaris Cluster media, again targeting the alternate boot environment.

Upgrading Nodes Using Oracle Solaris Zones

When upgrading a cluster node that uses the Solaris 10 OS, you need to consider the impact the upgrade process has on any Oracle Solaris Zones (also known as Oracle Solaris Containers) that are configured on the system. These containers represent additional, virtual Solaris instances that must also be upgraded.

To maintain service availability during an upgrade, a resource group must have a node list that contains at least one physical cluster node that will still be available to host the resource group while the rolling upgrade or dual-partition upgrade is performed. This is feasible both for resource groups that use global-cluster nonvoting nodes and for resource groups existing in zone clusters. However, HA containers are, by their very nature, the sole place in which the service resides. Therefore, taking the cluster node into single-user mode to patch it results in the HA container, and consequently the service, being unavailable for the duration of the update process [ZonePatch].

One way to minimize HA container service outage is to use Solaris Live Upgrade. The procedure is not particularly practical if you have tens or hundreds of HA containers because it involves many steps.

The following procedure is meant for environments where each HA container has its `zonepath`, on its own Oracle Solaris ZFS file system, in its own zpool, for example, `ora-pool/ora1` for `ora1`. Both `ora-pool` and `ora-pool/ora1` must have appropriate mount points set.

Solaris Live Upgrade requires full control over the zpools when the Oracle Solaris OS reboots in order to perform the necessary renaming of the ZFS file systems and the changes to the zones' configuration files. Therefore, the ZFS file systems must not be under Solaris Cluster software control during the shutdown and boot process. This can be achieved by creating an alternate boot environment where each zone and its associated `zonepath` are under cluster control. Then, prior to shutting down the system, you can offline and unmanage the resource groups controlling the HA containers and import each zpool that holds a `zonepath`. Finally, you can copy all the affected configuration files in the CCR and the `zpool.cache` file to the alternate boot environment.

In general, this procedure requires you to perform the following steps:

1. Alter the HA container's resource group node list to just the current node.
2. Perform a Solaris Live Upgrade on the node.
3. Disable the resources of the HA container's resource group, which takes the container offline and unmanages the resource groups.
4. Import the zpool holding the zone's `zonepath`.

5. Copy the `zpool.cache` file and corresponding CCR files for the resource group to the alternate boot environment.
6. Shut down and reboot the node.
7. After Solaris Live Upgrade has performed all of the necessary reconfigurations, manage and bring the resource group online.
8. Detach the HA container from its other nodes, and then use Solaris Live Upgrade and reboot the other nodes in turn.
9. Attach the `zonepath` of the HA container with the `-F` (force) option to bring the container's state back to `installed`.
10. Reset the HA container's resource group node list.
11. Check that switchover still works and finish by switching the HA container back to its primary machine.

Although this procedure requires a shorter outage, other factors might make it impractical. First, the HA container cannot be failed over while the upgrade is under way because the node list has been restricted to the single node. Additionally, the service-level agreements (SLAs) for other HA containers that are deployed on the system might limit when the outage can occur because any upgrade will affect them, too. If there isn't a single time period when groups of containers can be updated, then the restriction will prevent any of the changes from being made.

The final option for upgrading HA containers is to use the update on attach feature in the Solaris 10 10/08 release. This option can be used only to update the packages in a container and does not allow them to be downgraded. If you use this option, you must ensure that the process does not introduce any packages that are older than the current version the Solaris Cluster software requires. If it does, you must update them again using the `installer` program.

An outline of the steps for using the update on attach feature follows:

1. Evacuate (or switch over) all the containers that will not be patched from the target node (`nodeT`) using `clresourcegroup switch ...`, leaving just the containers to be patched.
2. On node `nodeT`, use `zoneadm -z ... detach` to detach each remaining container from `nodeT` so that the operating system will not attempt to patch them.
3. On node `nodeT`, patch `nodeT` and reboot back into the cluster.
4. Use `clresource disable ...` to disable the HA container resources that were switched over to the alternative node (`nodeA`), and switch their resource groups back to `nodeT`. This switches over the `zonepaths` to `nodeT`.

5. On node `nodeA`, detach the container from `nodeA` so that the operating system will not attempt to patch the container during the patch process or reboot the container after the upgrade.
6. On node `nodeT`, use `zoneadm -z ... attach -u` to reattach and patch the HA container on `nodeT`.
7. Test the containers by booting and halting them manually.
8. Place the containers back under cluster control by reenabling the HA container resource.
9. Repeat the upgrade process with `nodeA`.

Ensure that HA containers are prevented from failing between the nodes while this procedure is being followed.

Backing Up Your Cluster

Although your data might be protected by hardware RAID or host-based mirroring software and even possibly replicated to another site for disaster recovery purposes, you must have a consistent, usable backup of the data on your cluster. The requirement is twofold, involving backup of the root disk and backup of the application data. Both have their own specific challenges.

Root Disk Backup

Your root disk contains the Oracle Solaris OS with numerous configuration files that the system requires to perform its tasks. Not all of these files are static. Many of the log files you need to retain for auditing and debugging purposes are highly dynamic. Therefore, you must achieve a consistent backup of your system so that you can restore your system successfully, if the need arises.

When using UFS for the root disk, only two methods are available for achieving a guaranteed consistent backup of the root file system partitions:

- Boot the system into single-user mode.
- Use both `lockfs` and `fssnap` while the system is at its normal run level.

Obviously, booting a node into single-user mode requires that you switch over all the services hosted on this node. Not only does this result in service outages, but it also means that the application might have to share the resources on its new host node, which might degrade its performance somewhat. The `lockfs`/`fssnap` option

seems better. However, they can result in the system pausing while the data is flushed from the buffer cache and a consistent view is reached. If this pause is too long, it might have an adverse effect on the cluster framework. Furthermore, any real-time process prevents `fssnap` from being able to lock the file system. Thus, with a Solaris Cluster installation, you must temporarily suspend the `xntpd` daemon. However, other processes, such as the Oracle 10*g* Real Application Clusters or Oracle 11*g* Real Application Clusters frameworks might make this approach unworkable.

After you have performed the backup, you can delete the snapshot and move on to the next partition on the root disk.

Example 4.12 Using `lockfs` and `fssnap` to Create a Consistent Root (`/`) File System Snapshot

Stop the `xntpd` daemon before locking the root (`/`) file system with the `lockfs` command.

```
# /etc/rc2.d/S74xntpd.cluster stop
# lockfs -f
```

Take a snapshot of the root (`/`) file system using the `fssnap` command before restarting the `xntpd` daemon.

```
# time fssnap -o backing-store=/spare_disk /
/dev/fssnap/0

real    0m19.370s
user    0m0.003s
sys     0m0.454s
# /etc/rc2.d/S74xntpd.cluster start
Perform backup...
# fssnap -d /dev/fssnap/0
Deleted snapshot 0.
```

For an Oracle Solaris ZFS file system, the situation is much more straightforward. By issuing a `zfs snapshot` command, you can create a consistent view of a file system that you can back up and restore with confidence. Using the `-r` flag allows you to create these snapshots recursively for all file systems below a certain mount point, further simplifying the process.

Backing Up Application Data on a Cluster

The first challenge with backing up application data when a service resides on a cluster is determining which cluster node the service is currently running on. If a

failure has recently occurred, then the service might not be running on its primary node. If you are running Oracle RAC, the database is probably running on multiple nodes simultaneously. In addition, the data might be stored on raw disk or in Oracle's Automatic Storage Management (ASM), rather than in a file system. Consequently, any backup process must be capable of communicating with the node that currently hosts the application, rather than depending on the application being on a particular node, and potentially using application-specific backup procedures or software.

Although `fssnap` can be used in certain circumstances to achieve a consistent view of the root (`/`) file system partitions for backup, do not use it with failover UFS file systems. The pause in file system activity while the snapshot is being taken might result in the service fault probe detecting a fault and causing a service failover. Furthermore, `fssnap` cannot be used with global file systems (see the section "The Cluster File System" in Chapter 2, "Oracle Solaris Cluster: Features and Architecture") because `fssnap` must be run on the UFS mount point directly and works closely with the in-memory data structures of UFS. This means that the PxFS client and server (master) must interpret the `fssnap ioctl` system calls, but this capability is not currently present in PxFS.

Once more, the Oracle Solaris ZFS snapshot feature enables you to obtain a consistent view of the application data and so is a simpler option if there are no specific tools for consistently backing up the application data.

Many backup products are available from Oracle and from third-party sources. Many have application-specific integration features, for example, the ability to integrate with Oracle's RMAN backup function. Most products can back up data stored in any file system (UFS, ZFS, QFS, VxFS) that you might have configured in your cluster.

Highly Available Backup Servers

It's obviously very important to perform regular, secure backups of your critical systems. This, in turn, means that the systems performing the backup must be sufficiently highly available. Otherwise, they might not be able to complete a backup within the time window available. Although there is little you can do to make an individual tape drive more available, you can have tape libraries housing multiple tape drives. Then the problem of availability rests with the system that controls the backups.

A backup (master) server contains the backup configuration information: catalogs of previous backups, schedules for subsequent backups, and target nodes to be backed up. Just like any other service, this collection of data files and the programs that access it can be made highly available. Thus, a highly available service can be achieved by placing the configuration files on a highly available file system,

hosted by one or more Solaris Cluster nodes, and encapsulating the backup server program in a suitable resource in a resource group.

The most common data center backup configuration uses SAN-attached tape libraries with multiple tape drives. You configure the master server to manage the backup by communicating with the client software installed on each target cluster node to be backed up. Instead of defining an entire physical server as a target, you use the logical host of the individual services that require their data to be backed up. The master server then contacts the appropriate physical node when the time comes to back up the data. If you need to back up the individual nodes, then you define the backup so that it covers only the file systems that constitute the root (`/`) file system. When the time comes to perform the backup, the master server directs the client to stream the necessary dataset to one or more tapes in the library.

Solaris Cluster agents are available for both the StorageTek Enterprise Backup software and Veritas NetBackup. If a Solaris Cluster agent is not available for your backup software, you can easily create one, as described in the next section.

Creating New Resource Types

As described in the section "Data Service and Application Agents" in Chapter 2, "Oracle Solaris Cluster: Features and Architecture," Oracle has a substantial list of supported agents that cover most of the applications in your data center. These application agents are maintained by Oracle and are extensively tested on each new release of both the Solaris Cluster software and the application itself. Even so, inevitably you will have an application that is not part of the existing agent portfolio.

Application Suitability

Before creating a resource type for your application, you must determine whether the application meets the criteria for being made highly available. The following list highlights the main points you must consider. For a complete list see "Analyzing the Application for Suitability" in [SCDevGuide].

- Is your application crash-tolerant? This is important because in a highly available environment your application must be able to recover its data consistency without requiring manual intervention. If the application did require such intervention, then most of the benefits of a high-availability framework would be lost.
- Does your application rely on the physical node name of the machine, such as that resulting from calls to `uname`, `gethostbyname`, or equivalent interfaces?

If so, then when the application moves to another cluster node, the dependency on the physical hostname will probably cause the application to fail. There is a work-around to this problem, which is to interpose the `libschost.so.1` library. However, this work-around can sometimes raise support issues with application vendors.

- Can your application run on a multihomed system, that is, one with several public networks? Your application must be able to handle situations where IP addresses are configured and unconfigured from network adapters as services move around the cluster. This has consequences for the way your application binds to the network.
- Does your application use hard-coded path names for the location of its data? If so, then symbolic links might not be sufficient to ensure that the data is stored in a location that is compatible with using a failover or global file system. If the application renames a data file, it can break the symbolic links.

After you have determined that your application is suitable for being made highly available, you have several ways to achieve the necessary integration:

- You can use the Generic Data Service (GDS) directly and just supply the required parameters. Although you cannot define any new extension properties for the resource type you create, it is by far the simplest option.
- You can create a subclass of the GDS to create a completely new resource type. This option enables you to define one or more extension properties for your new resource type. This option is relatively simple and yet provides considerable flexibility.
- You can extend the GDS using the Advanced Agent Toolkit. Although this option does not create a new resource type, it does enable you to define one or more extension properties. This option is also relatively simple and provides considerable flexibility.
- You can use the GUI `scdsbuilder` tool and customize the resulting shell script or C source using the Resource Management API (RMAPI) and the Data Service Development Library (DSDL) APIs. If significant customization work is needed, this option might result in an increased maintenance burden.
- You can use the RMAPI or DSDL APIs directly to develop your resource type from scratch. This option trades the development and maintenance costs for ultimate flexibility and performance.

Each option is discussed in more detail in the following sections.

Generic Data Service

The Generic Data Service (GDS) is provided with the Solaris Cluster software. The `SUNW.gds` agent is packaged in the `SUNWscgds` package, which is installed as standard by the Solaris Cluster software `installer` program. The `SUNW.gds` agent is considered the preferred way to create both failover and scalable resources. The GDS is supported by Oracle, but you must support the script that you provide for the `Start_command`, `Stop_command`, `Probe_command`, and `Validate_command` methods.

By default, the `SUNW.gds` resource type is not registered, so you must register it before attempting to create a resource of that type. The commands in the following example show how to determine if the resource type is registered and then how to register it, if it is not already present.

Example 4.13 Registering the `SUNW.gds` Resource Type

Use the `clresourcetype` command to determine whether the `SUNW.gds` resource type needs to be registered.

```
# clresourcetype list | grep SUNW.gds
# clresourcetype register SUNW.gds
# clresourcetype list | grep SUNW.gds
SUNW.gds:6
```

In addition to the standard resource properties, the GDS agent has four properties to enable you to integrate your application: `Start_command`, `Stop_command`, `Probe_command`, and `Validate_command`. These properties are described in "Integrating Your Application-Specific Logic." By using the GDS as the basis for your application, you automatically benefit from all the patches and feature upgrades that the GDS receives.

Example 4.14 shows how you can use the GDS to make the X11 program `xeyes` highly available. You begin by creating a `Start_command` program. In this example, a script calls the full path name of the program with a parameter that is passed to the shell script. This script must exist on all the cluster nodes on which the application is intended to run.

Next, having checked that the `SUNW.gds` resource type is registered, you create the resource group. In this example, you allow the resource group's node list to default to all the cluster nodes.

Next, you create a resource to represent your program. In the example, the `Start_command` property is specified by the script you wrote (and which must exist on all nodes). The display parameter to use is also specified. Because this

program does not listen on any network ports, you set the `network_aware` property to `false`. This means that the probe mechanism used will be the continued existence of the `xeyes` process that the `Start_command` program leaves running in the background. By default, any resource you create is enabled so that when the resource group is brought online, the resource is automatically started. To change the default, you can specify the `-d` argument to the `clresource create` command.

The last two steps instruct the RGM that it needs to control or manage the `xeyes-rg` resource group and then to bring that resource group online. The action of bringing the resource group online starts the resource because it was created in an enabled state.

Assuming you have allowed remote X11 clients to display on your X server using `xhost` and you have specified the correct X display to use (substitute a value suited to your environment for `myhost:1.0`), then the `xeyes` program will appear on your display. You can switch the resource group between nodes and the RGM will kill the `xeyes` process and restart it on the new node, `phys-summer2`, as the example shows.

Example 4.14 Creating a Simple, Highly Available `xeyes` Service

List the script that will be used to start the `xeyes` command.

```
# cat /tmp/start_xeyes
#!/bin/ksh
/usr/openwin/demo/xeyes -display $1 &
exit 0
```

Check that the `SUNW.gds` resource type is registered, and then create the resource group and resource that will control the `xeyes` service.

```
# clresourcetype list | grep SUNW.gds
SUNW.gds:6
# clresourcegroup create xeyes-rg
# clresource create -t SUNW.gds \
> -p start_command="/tmp/start_xeyes myhost:1.0" \
> -p network_aware=false \
> -g xeyes-rg xeyes-rs
```

Use the `clresourcegroup` command to bring the `xeyes-rg` resource group online.

```
# clresourcegroup manage xeyes-rg
# clresourcegroup online xeyes-rg
# clresourcegroup status xeyes-rg

=== Cluster Resource Groups ===

Group Name       Node Name       Suspended      Status
----------       ---------       ---------      ------
xeyes-rg         phys-summer1    No             Online
                 phys-summer2    No             Offline
```

```
# clresourcegroup switch -n phys-summer2 xeyes-rg
# clresourcegroup status xeyes-rg

=== Cluster Resource Groups ===

Group Name       Node Name       Suspended      Status
----------       ---------       ---------      ------
xeyes-rg         phys-summer1    No             Offline
                 phys-summer2    No             Online
```

To demonstrate how the GDS handles application failure, quit the `xeyes` program from your X display. You will notice that the RGM restarts the application almost instantaneously. The messages in `/var/adm/messages` (see Example 4.15) indicate that the RGM recognized the failure and restarted the service.

After the fault probe determines that the service is online, indicated by `Service is online` in `/var/adm/messages`, kill the process again. The resource has two properties that determine how many times it is restarted by the RGM within a certain time period. These properties are `Retry_count` and `Retry_interval` (see Example 4.16). After the specified number of failures, the built-in logic of the GDS determines that the current node is unhealthy and releases the service so that it can be started on another node. If the service also experiences problems on this node, then the RGM will not fail the service back to its original node unless the time period, in seconds, as defined by the resource group's `Pingpong_interval` property, has passed. Instead, the GDS attempts to keep the service running on the remaining node. This behavior is governed by another property called `Failover_mode`.

The purpose of the `Pingpong_interval` property is to prevent a service that fails to start from endlessly looping, resulting in the service migrating back and forth between cluster nodes. In a test environment, you might need to reset the value of `Pingpong_interval` to a lower value. Doing so enables you to restart your service once you have corrected any problems you encountered.

Example 4.15 Sample RGM Messages

The `/var/adm/messages` file contains information on the state changes of the resource groups and resources in the cluster.

```
Nov 23 04:00:23 phys-summer2 Cluster.RGM.global.rgmd: [ID 529407 daemon.notice]
resource group xeyes-rg state on node phys-summer2 change to RG_ONLINE
Nov 23 04:01:23 phys-summer2 Cluster.RGM.global.rgmd: [ID 922363 daemon.notice]
resource xeyes-rs status msg on node phys-summer2 change to <Service is online.>
Nov 23 04:01:25 phys-summer2 Cluster.PMF.pmfd: [ID 887656 daemon.notice] Process:
tag="xeyes-rg,xeyes-rs,0.svc", cmd="/bin/sh -c /tmp/start_xeyes myhost:1.0", Failed
to stay up.
```

```
Nov 23 04:01:25 phys-summer2 Cluster.RGM.global.rgmd: [ID 784560 daemon.notice]
resource xeyes-rs status on node phys-summer2 change to R_FM_FAULTED
Nov 23 04:01:25 phys-summer2 Cluster.RGM.global.rgmd: [ID 922363 daemon.notice]
resource xeyes-rs status msg on node phys-summer2 change to <Service daemon not
running.>
Nov 23 04:01:25 phys-summer2 SC[,SUNW.gds:6,xeyes-rg,xeyes-rs,gds_probe]: [ID 423137
daemon.error] A resource restart attempt on resource xeyes-rs in resource group
xeyes-rg has been blocked because the number of restarts within the past Retry_
interval (370 seconds) would exceed Retry_count (2)
Nov 23 04:01:25 phys-summer2 SC[,SUNW.gds:6,xeyes-rg,xeyes-rs,gds_probe]: [ID 874133
daemon.notice] Issuing a failover request because the application exited.
Nov 23 04:01:25 phys-summer2 Cluster.RGM.global.rgmd: [ID 494478 daemon.notice]
resource xeyes-rs in resource group xeyes-rg has requested failover of the resource
group on phys-summer2.
Nov 23 04:01:25 phys-summer2 Cluster.RGM.global.rgmd: [ID 423291 daemon.error] RGM
isn't failing resource group <xeyes-rg> off of node <phys-summer2>, because there are
no other current or potential masters
Nov 23 04:01:25 phys-summer2 Cluster.RGM.global.rgmd: [ID 702911 daemon.error]
Resource <xeyes-rs> of Resource Group <xeyes-rg> failed pingpong check on node <phys-
summer1>.  The resource group will not be mastered by that node.
Nov 23 04:01:25 phys-summer2 SC[,SUNW.gds:6,xeyes-rg,xeyes-rs,gds_probe]: [ID 969827
daemon.error] Failover attempt has failed.
Nov 23 04:01:25 phys-summer2 SC[,SUNW.gds:6,xeyes-rg,xeyes-rs,gds_probe]: [ID 670283
daemon.notice] Issuing a resource restart request because the application exited.
```

Example 4.16 Retry, Failover Mode, and Ping-pong Interval Properties

Use the `clresource` command to determine the property values of the `xeyes-rs` resource.

```
# clresource show \
>    -p retry_count,retry_interval,failover_mode xeyes-rs

=== Resources ===

Resource:                                       xeyes-rs

  --- Standard and extension properties ---

  Retry_interval:                               370
    Class:                                         standard
    Description:                                   Time in which monitor attempts to
restart a failed resource Retry_count times.
    Type:                                          int

  Retry_count:                                  2
    Class:                                         standard
    Description:                                   Indicates the number of times a
monitor restarts the resource if it fails.
    Type:                                          int

  Failover_mode:                                SOFT
    Class:                                         standard
    Description:                                   Modifies recovery actions taken
when the resource fails.
    Type:                                          enum
```

```
# clresourcegroup show -p pingpong_interval xeyes-rg

=== Resource Groups and Resources ===

Resource Group:                                 xeyes-rg
  Pingpong_interval:                               3600
```

In the preceding example, the display variable property can be changed only by stopping the resource and modifying the `Start_command` property. Although of little importance here, because the `xeyes` program must be restarted to change the target X server on which it displays, it does make a difference in instances where a variable can be changed while a service is running. Examples include changing debugging levels to use and changing directories for log files.

To create a resource type that has new extension properties that can be changed when you need to change them, you need to either write your resource type from scratch or create a subclass of the GDS, as described in a later section.

Supporting New Applications Using the Advanced Agent Toolkit

Many application agents in the current Solaris Cluster software release are derived from the Advanced Agent Toolkit methodology [AdvGDSTlkit]: HA-PostgreSQL, HA-MySQL, and HA containers, to name three. All three use the `SUNW.gds` agent as their basis. However, in its raw form, the `SUNW.gds` agent has some limitations.

The rationale behind the toolkit is that all new application agents have many common requirements:

- They might require one or more extension properties.
- They must provide debugging information.
- They might need to disable the process-monitoring facility (`pmfadm`) for applications that leave no obvious child processes to monitor.
- They must supply a `Start_command` script, as a minimum, and possibly `Stop_command`, `Probe_command`, and `Validate_command` scripts.

The toolkit also simplifies much of the work needed to handle Oracle Solaris Zones and SMF. Thus, providing this extended framework enables your developers to focus on the application-specific integration work rather than on debugging the framework itself. After the work is complete, the new resource type is registered using a registration script.

Developing Resource Types by Creating a Subclass of the GDS

The advantage of creating a subclass of the GDS, rather than writing a new resource type from scratch, is that the new resource type inherits all the best practices that are already part of the standard GDS code. In addition, creating a subclass of the GDS enables you to create your own resource type extension properties while retaining the same level of flexibility as if you had started from scratch. Finally, your new resource type, which is a subclass of the GDS, has a distinct name, enabling you to easily distinguish resources of the new resource type. If you instead used the Advanced Agent Toolkit or the `SUNW.gds` agent, then you would have to determine what the resource is by examining the extension properties or reviewing the code. This step would be necessary because the resource type would be set to `SUNW.gds`, rather than `MYCORP.appsvr`, for example.

You create a subclass of the GDS by creating a resource type registration (RTR) file where the `RT_basedir` parameter is set to the directory containing binaries used by the standard GDS methods: `Start`, `Stop`, `Validate`, and so on. You then extend the RTR file by defining your own resource type extension properties. Finally, you set the method parameters in the RTR file to point to your scripts that override the standard GDS behavior.

Several existing Sun resource types are implemented this way, including the HA-Logical Domain agent (`SUNW.ldom`), which was covered in the section "Failover Guest Domains" in Chapter 3, "Combining Virtualization Technologies with Oracle Solaris Cluster Software."

The RTR file for the `SUNW.ldom` resource type is shown in Example 4.17. In this RTR file, the `RT_basedir` parameter is set to the standard directory for the GDS package, that is, `/opt/SUNWscgds/bin`. Of the standard methods, only `Init`, `Boot`, and `Validate` have been overridden using programs that are located in the `../../SUNWscxvm/bin` directory. Unlike a standard GDS resource type, the `Start_command`, `Stop_command`, `Probe_command`, and `Validate_command` properties are assigned fixed values and cannot be changed. This is indicated by the `Tunable = NONE` settings. Furthermore, each command, apart from `validate_command`, is called with a consistent set of arguments, namely, `-R %RS_NAME -T %RT_NAME -G %RG_NAME`. The `%variable` construct is similar to the `$variable` syntax found in shell scripts. It means that when a resource of this type is instantiated, use the names you assigned it as arguments. For example, if you wrote a resource type called `FOO.bar` and then created a resource group called `whizz-rg` containing a resource called `bang-rs` of this type, the argument passed would be `-R bang-rs -T FOO.bar -G whizz-rg`. With these arguments, you can then make calls to the RMAPI or DSDL APIs to retrieve or set properties.

In contrast to the `Start_command`, `Stop_command`, and `Probe_command` properties, the `Validate_command` property does not use this construct. Instead, the

RGM passes the validate command all the properties listed for the resource type on the command line. Then the validate command parses this list and determines whether the configuration is valid.

Example 4.17 RTR File for the `SUNW.ldom` Resource Type

The following text shows some of the key parts of the `RTR` file for the `SUNW.ldom` resource type:

```
      .
      .
      .
RESOURCE_TYPE = "ldom";
VENDOR_ID = SUNW;
RT_DESCRIPTION = "Sun Cluster HA for xVM Server SPARC Guest Domains";

RT_version ="1";
API_version = 10;

RT_basedir=/opt/SUNWscgds/bin;

Init                            =       ../../SUNWscxvm/bin/init_xvm;
Boot                            =       ../../SUNWscxvm/bin/boot_xvm;

Start                           =       gds_svc_start;
Stop                            =       gds_svc_stop;

Validate                        =       ../../SUNWscxvm/bin/validate_xvm;
Update                          =       gds_update;

Monitor_start                   =       gds_monitor_start;
Monitor_stop                    =       gds_monitor_stop;
Monitor_check                   =       gds_monitor_check;

Init_nodes = RG_PRIMARIES;
Failover = FALSE;

# The paramtable is a list of bracketed resource property declarations
# that come after the resource-type declarations
# The property-name declaration must be the first attribute
# after the open curly of a paramtable entry
#
# The following are the system defined properties. Each of the system defined
# properties have a default value set for each of the attributes. Look at
# man rt_reg(4) for a detailed explanation.
#
{
        PROPERTY = Start_timeout;
        MIN = 60;
        DEFAULT = 300;
}
{
        PROPERTY = Stop_timeout;
        MIN = 60;
        DEFAULT = 300;
}
      .
      .
      .
```

```
# This is an optional property. Any value provided will be used as
# the absolute path to a command to invoke to validate the application.
# If no value is provided, The validation will be skipped.
#
{
        PROPERTY = Validate_command;
        EXTENSION;
        STRING;
        DEFAULT = "";
        TUNABLE = NONE;
        DESCRIPTION = "Command to validate the  application";
}

# This property must be specified, since this is the only mechanism
# that indicates how to start the application.  Since a value must
# be provided, there is no default.  The value must be an absolute path.
{
        PROPERTY = Start_command;
        EXTENSION;
        STRINGARRAY;
        DEFAULT = "/opt/SUNWscxvm/bin/control_xvm start -R %RS_NAME -T %RT_NAME -G
          %RG_NAME";
        TUNABLE = NONE;
        DESCRIPTION = "Command to start application";
}

# This is an optional property.  Any value provided will be used as
# the absolute path to a command to invoke to stop the application.
# If no value is provided, signals will be used to stop the application.
#
# It is assumed that Stop_command will not return until the
# application has been stopped.
{
        PROPERTY = Stop_command;
        EXTENSION;
        STRING;
        DEFAULT = "/opt/SUNWscxvm/bin/control_xvm stop -R %RS_NAME -T %RT_NAME -G
          %RG_NAME";
        TUNABLE = NONE;
        DESCRIPTION = "Command to stop application";
}

# This is an optional property.  Any value provided will be used as
# the absolute path to a command to invoke to probe the application.
# If no value is provided, the "simple_probe" will be used to probe
# the application.
#
{
        PROPERTY = Probe_command;
        EXTENSION;
        STRING;
        DEFAULT = "/opt/SUNWscxvm/bin/control_xvm probe -R %RS_NAME -G %RG_NAME -T
          %RT_NAME";
        TUNABLE = NONE;
        DESCRIPTION = "Command to probe application";
}

# This is an optional property.  It determines whether the application
# uses network to communicate with its clients.
#
{
        PROPERTY = Network_aware;
        EXTENSION;
        BOOLEAN;
        DEFAULT = FALSE;
```

```
        TUNABLE = AT_CREATION;
        DESCRIPTION = "Determines whether the application uses network";
}

# This is an optional property, which determines the signal sent to the
# application for being stopped.
#
{
        PROPERTY = Stop_signal;
        EXTENSION;
        INT;
        MIN = 1;
        MAX = 37;
        DEFAULT = 15;
        TUNABLE = WHEN_DISABLED;
        DESCRIPTION = "The signal sent to the application for being stopped";
}

# This is an optional property, which determines whether to failover when
# retry_count is exceeded during retry_interval.
#
{
        PROPERTY = Failover_enabled;
        EXTENSION;
        BOOLEAN;
        DEFAULT = TRUE;
        TUNABLE = WHEN_DISABLED;
        DESCRIPTION = "Determines whether to failover when retry_count is exceeded
during retry_interval";
}

# This is an optional property that specifies the log level GDS events.
#
{
        PROPERTY = Log_level;
        EXTENSION;
        ENUM { NONE, INFO, ERR };
        DEFAULT = "INFO";
        TUNABLE = ANYTIME;
        DESCRIPTION = "Determines the log level for event based traces";
}

{
        Property = Debug_level;
        Extension;
        Per_node;
        Int;
        Min = 0;
        Max = 2;
        Default = 0;
        Tunable = ANYTIME;
        Description = "Debug level";
}

{
        Property = Domain_name;
        Extension;
        String;
        Minlength = 1;
        Tunable = WHEN_DISABLED;
        Description = "LDoms Guest Domain name";
}
```

```
{
        Property = Migration_type;
        Extension;
        Enum { NORMAL, MIGRATE };
        Default = "MIGRATE";
        Tunable = ANYTIME;
        Description = "Type of guest domain migration to be performed";
}

{
        PROPERTY = Plugin_probe;
        EXTENSION;
        STRING;
        DEFAULT = "";
        TUNABLE = ANYTIME;
        DESCRIPTION = "Script or command to check the guest domain";
}

{
        PROPERTY = Password_file;
        EXTENSION;
        STRING;
        DEFAULT = "";
        TUNABLE = WHEN_DISABLED;
        DESCRIPTION = "The complete path to the file containing the target host
password";
}
```

`scdsbuilder` GUI

To customize an agent beyond what is permitted by the GDS, you can use the Agent Builder command, `scdsbuilder` (see the `scdsbuilder`(1HA) man page). This command has three code generation options, and the resulting files are wrapped in a Solaris package that you can install on your cluster nodes:

- DSDL code (see the section "Data Service Development Library").
- `ksh` code, including all the necessary `scha_control` commands (see the section "Resource Management API"). With the `ksh` code, you are creating your own resource type.
- A `ksh` registration script for a GDS agent. Here, the code generates the appropriate `clresource create` command.

You can customize the resulting code to your specific needs. However, with the `ksh` registration script for the GDS agent, the scope for modification is limited. The example in Figure 4.7 shows the use of the third option.

The `scdsbuilder` command starts the Solaris Cluster Agent Builder GUI, as shown in Figure 4.7. In this example, data has already been specified for each field available to the user. A short code of `SUNW` is specified for the vendor name, and `tstgds` is specified for the application name. This data is then used to generate

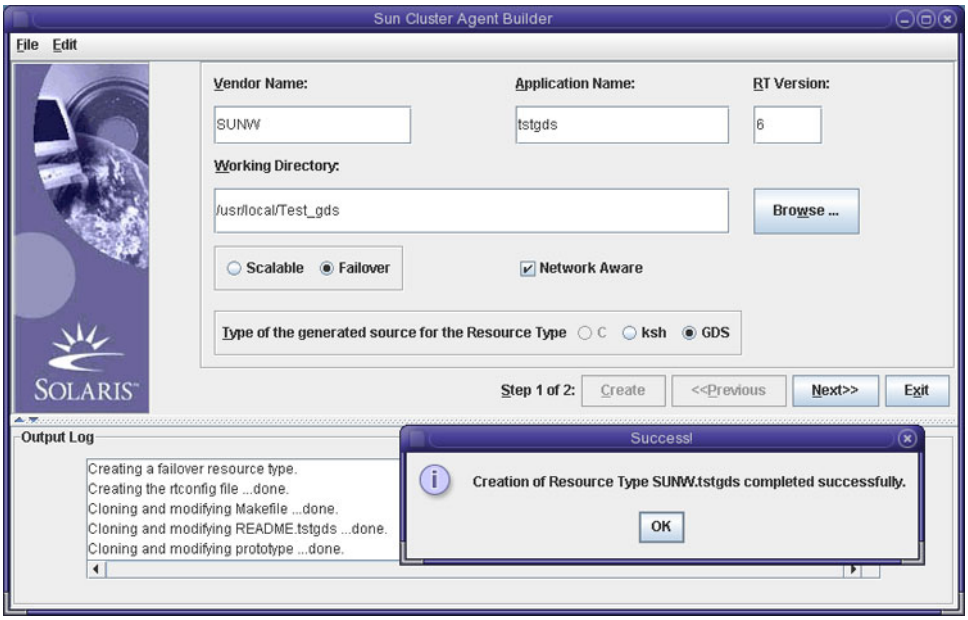

Figure 4.7 Using the `scdsbuilder` GUI to create a new resource type

both the name of the package that Agent Builder creates for you and the name of the resource type that you will subsequently use.

The information you provide in the other fields is used as follows:

- The RT version enables you to specify a version number for this resource type. You can identify which version of the agent you are running when it is placed into production.
- The working directory is used by Agent Builder as a working area in which it can create your package and write other associated, intermediate files.
- Your target application determines whether you select the scalable or failover option. If a particular instance of an application can run on multiple nodes at once without corrupting any of its data files, then you can select the scalable option. A good example of such an application is a web server. For all other applications, such as databases and file services, select the failover option.
- The Network Aware check box is used to determine whether any resource created using this resource type needs to have the `port_list` property set. The `port_list` property is then used by the GDS service to provide a simple probe mechanism.

- The source type option determines whether the resulting code uses the C programming language, `ksh`, or the GDS (see the section "`SUNW.gds`" in Chapter 2, "Oracle Solaris Cluster: Features and Architecture") to create the data service. To use the C option, you must have a C compiler installed on your system.

After you have entered the data and clicked on the Next button, you are presented with the screen shown in Figure 4.8.

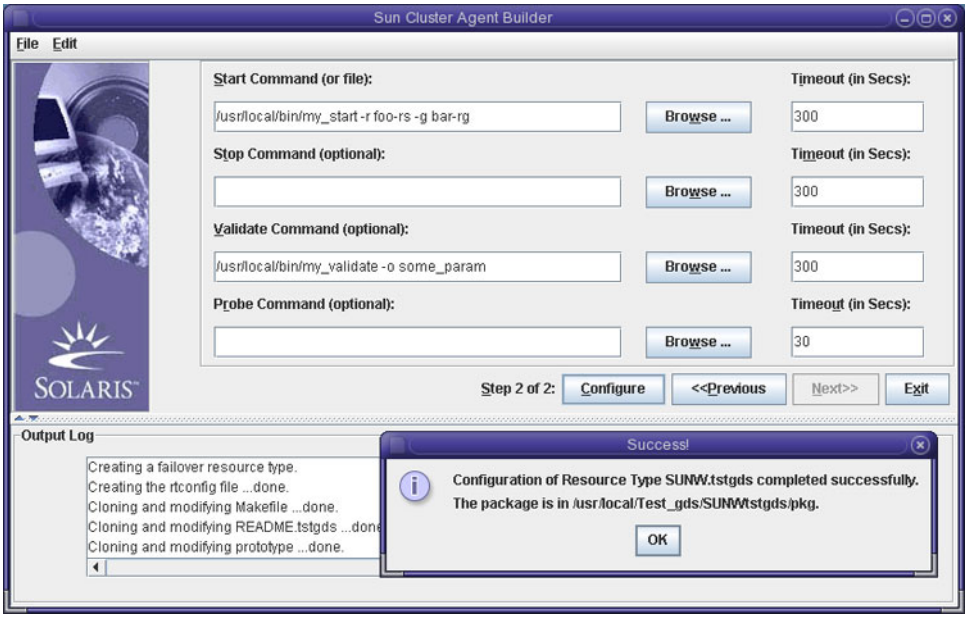

Figure 4.8 Completing the resource type definition using `scdsbuilder`

Integrating Your Application-Specific Logic

You use the fields in this second screen to provide the location of the programs (which can be compiled executables or scripts) and their associated arguments that will be used to start, stop, probe, and validate your data service when it is installed on the target cluster nodes. For each program, you can set a time limit on how long it can take for the program to complete. If the program does not complete within the allocated time period, then the resource is placed into a failed state, such as `STOP_FAILED`.

You are required to provide a value only for the start program. All the other programs are optional. Any programs specified must exit with a return code of zero

only when they have successfully completed their work. If they fail to perform their allotted task, they must return a value greater than 100. Values below that are used by the Solaris Cluster commands and have specific meanings (see the `intro`(1CL) man page).

The programs you assign to the start and stop commands must return successfully only when your target application has actually completed the relevant operation. If the stop command leaves the application under its control running, or not completely stopped, but the stop command returns successfully, then the cluster framework erroneously determines that it is safe to start the resource group on another cluster node. In some instances, particularly when the application uses a global file system, this outcome could result in data corruption because the two instances of the application could write to their data files in an uncontrolled fashion.

If no stop command is provided, the process tree that results from the start command is terminated using the `kill` command.

The validate command enables you to check that your application is correctly configured on all the potential nodes on which it can run. Again, if the program determines that your application is misconfigured, the validate program must exit with a nonzero exit code.

The capability to incorporate a probe command is one of the key benefits of using the Solaris Cluster framework. A probe command enables you to write a program that determines the health of your application. As an example, if you are writing a probe for a database, you could test whether it can execute basic SQL statements, such as creating or deleting a table, or adding or deleting a record. If you do not provide a probe script, then default methods are used instead.

For non-network-aware applications, the process-monitoring command `pmfadm` (see the `pmfadm`(1M) man page) monitors the process tree spawned by your start command. Only if all the processes have failed will the cluster framework attempt to restart the service. Therefore, if your service consists of multiple processes and only one process fails, then `pmfadm` will not recognize this fault unless it causes all the other processes to fail as well. Consequently, if you need to monitor your application with a higher degree of granularity, you must provide a custom fault probe.

If the application is network-aware, then the default probe tries to open the port listed in the `port_list` property. Because this is a simple probe, it makes no attempt to retrieve any data. Even if the default probe successfully opens the ports, that does not necessarily indicate overall application health.

In the preceding example, you would install the package generated by `scdsbuilder` on all your cluster nodes. You would then register the new resource type so that you could create new resources of this type. When the RGM is requested to create a resource, it calls the validate command: `/usr/local/bin/my_validate -o some_param`. If that command succeeds and you enable the resource, the RGM calls the `/usr/local/bin/my_start -r foo-rs -g bar-rg` command. In both

cases, the initial arguments are fixed, but you can modify them subsequently using the `clresource` command.

Resource Type Registration File

If you decide to write an agent from scratch using either the RMAPI or DSDL APIs, you must first describe the properties of your proposed resource type in a file known as the resource type registration (RTR) file. This file provides the RGM with details on which programs to call and which variables are required to control the particular application.

Example 4.18 shows an extract from the `SUNW.LogicalHostname` RTR file. As the example shows, all the programs for this resource type are located in the directory defined by `RT_BASEDIR`. The RTR file also defines programs that will, among other tasks, start, stop, and probe (`Monitor_start`) the logical IP address that the resource plumbs. These addresses are, in turn, defined in the `HostnameList` property.

The extension properties you define are all application-specific. They could, for example, refer to the location of the software binaries, that is, the application home directory. If a property has a default value, then you can define it in the RTR file to save your system administrator from having to override it each time he or she creates a resource of this type. Furthermore, you can place limits on what values certain properties can take and when they can be changed.

Example 4.18 Extract from the `SUNW.LogicalHostname` RTR File

The following text shows some of the key parts of the RTR file for the `SUNW.LogicalHostname` resource type:

```
#
# Copyright 1998-2008 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#

#ident "@(#)SUNW.LogicalHostname 1.20 08/05/20 SMI"

# Registration information and Paramtable for HA Failover IPaddress
#
# NOTE: Keywords are case insensitive, i.e. users may use any
# capitalization style they wish
#

RESOURCE_TYPE ="LogicalHostname";
VENDOR_ID = SUNW;
RT_DESCRIPTION = "Logical Hostname Resource Type";
```

```
SYSDEFINED_TYPE = LOGICAL_HOSTNAME;

RT_VERSION ="3";
API_VERSION = 2;

INIT_NODES = RG_PRIMARIES;

RT_BASEDIR=/usr/cluster/lib/rgm/rt/hafoip;

FAILOVER = TRUE;

# To enable Global_zone_override
GLOBAL_ZONE = TRUE;

START                           =       hafoip_start;
STOP                            =       hafoip_stop;

PRENET_START                    =       hafoip_prenet_start;

VALIDATE                        =       hafoip_validate;
UPDATE                          =       hafoip_update;

MONITOR_START                   =       hafoip_monitor_start;
MONITOR_STOP                    =       hafoip_monitor_stop;
MONITOR_CHECK                   =       hafoip_monitor_check;

PKGLIST = SUNWscu;

#
# Upgrade directives
#
#$upgrade
#$upgrade_from "1.0" anytime
#$upgrade_from "2" anytime

# The paramtable is a list of bracketed resource property declarations
# that come after the resource-type declarations
# The property-name declaration must be the first attribute
# after the open curly of a paramtable entry
#
# The Paramtable cannot contain TIMEOUT properties for methods
# that aren't in the RT
{
        PROPERTY = Start_timeout;
        MIN=360;
        DEFAULT=500;
}
      .
      .
      .
# HostnameList: List of hostnames managed by this resource. All must be
# on the same subnet. If need > 1 subnet with a RG, create as many
# resources as there are subnets.
{
        PROPERTY = HostnameList;
        EXTENSION;
        STRINGARRAY;
        TUNABLE = AT_CREATION;
        DESCRIPTION = "List of hostnames this resource manages";
}
      .
      .
      .
```

Resource Management API

The Resource Management API (RMAPI) is a set of low-level functions contained in the `libscha.so` library with both C and shell interfaces. All the function names provided by this interface are prefixed with `scha_`. The shell interfaces are listed in section 1HA of the Solaris Cluster manual pages.

The `ksh` scripts generated by the Agent Builder are built using these commands, so you can insert additional lines in this code where the comments indicate. However, for greater control over the logic imposed on your application you must write your application agent from scratch.

Data Service Development Library

The Data Service Development Library (DSDL) is a set of higher-level functions encapsulated in the `libdsdev.so` library that builds on the RMAPI functionality. This library can only be accessed using a C programming language interface. Consequently, it is potentially more time-consuming to write a complete application agent using this approach, although it does offer the greatest level of performance and flexibility.

If you used Agent Builder to create a resource type, you can customize it by inserting extra DSDL code where the comments indicate. Otherwise, you must write your agent from scratch.

All the function names provided by the library are prefixed with `scds_` and are documented in section 3HA of the Solaris Cluster manual pages. The NFS agent source code [NFSAgent] serves as a good example of how these APIs are used. Using the `nfs_svc_start.c` source as a specific example, the library is initialized with `scds_initialize()`. Resource and resource group names are then retrieved using `scds_get_resource_name()` and `scds_get_resource_group_name()` calls, respectively. Finally, the status of the resource is set by the RMAPI `scha_resource_setstatus()` call. Most of the coding effort involved with using these interfaces is consumed by the logic that describes how the agent should behave in various failure scenarios. For example, how many times should the agent attempt to restart the service before giving up and potentially failing over? What should the agent do in response to a network failure?

One advantage of using the GDS is that all the best practices for service behavior are already in the logic of the code that makes up the agent, saving you from re-creating that code.

Useful Utilities for Building Custom Data Services

The Solaris Cluster software comes with two programs that you will find very useful if you create your resource type from scratch: `hatimerun` (see the `hatimerun`(1M) man page) and `pmfadm`.

`hatimerun` Command

Throughout the `Start`, `Stop`, `Monitor_start`, and `Validate` methods of your resource type, you will need to run various programs to perform the required logic steps. Because your goal is high availability, you cannot wait for a program that might never respond or return, whether that program has gone into a loop or is unable to retrieve some important data from the network, disk, or other program. Consequently, you must place time constraints on the duration of the program's execution. This is the function of the `hatimerun` command. It enables you to execute a program under its control and set a limit on the time it can take to respond. If the program in question fails to respond in a timely fashion, it is terminated by default.

The `hatimerun` command also enables you to leave the program running asynchronously in the background, change the exit code returned after a timeout, or use a particular signal to terminate your program.

The most common usage of this command is in your probe commands or in the steps leading up to stopping or starting your application.

`pmfadm` Command

If you write a custom probe for your service, you decide what constitutes a healthy service. The criteria might include application-specific checks to determine if the data it is delivering to potential clients is valid or timely. If the application consists of multiple processes, you might want to check that each process is running, using the `ps` command. All of these tests combine to give you the best assessment of your application's current health. However, your probe is scheduled to make its checks only at regular intervals. Even though you can tune these checks to occur at shorter intervals, doing so results in a greater load on your system. Consequently, you must wait, on average, half the probe period before your probe detects a situation where your application has completely failed, meaning that all the processes have exited. Once again, this does not help much toward your goal of high availability.

The solution is to use `pmfadm`, the process-monitoring facility command. When you start your application under `pmfadm`, it monitors all the processes your application spawns to a level that you determine. By default, it monitors all the application's child processes. If they all exit, `pmfadm` immediately restarts your

application for you on the condition that it has not already exceeded a preset number of restarts within a certain time interval.

The most common usage of this command is in your start command to ensure that your key application processes are monitored and that complete failures are reacted to immediately.

`libschost.so` Library

Some applications store or make use of configuration information about the physical hostname of the server on which the application is running. Such applications will most likely fail when the application is placed in a resource group and moved between the nodes of a cluster. This failure occurs because calls to `uname` or `gethostbyname` produce different responses on the global zone of each cluster node. Oracle Application Server and the Oracle E-Business Suite are two examples of programs that risk such failures [LibHost].

To overcome this limitation, you use the `LD_PRELOAD` feature to enable the runtime linker to interpose the `libschost.so.1` library in the dynamic linking process. The following example shows how this is done. You can use the same construct within your resource `Start` or `Monitor_start` (probe) methods, as required.

Example 4.19 How to Use the `sclibhost.so.1` Library to Change the String Returned as the Hostname

Use the `uname` command to display the current hostname.

```
# uname -n
phys-winter1
```

Set the `LD_PRELOAD_32`, `LD_PRELOAD_64` and `SC_LHOSTNAME` environment variables, and then rerun the `uname` command.

```
# LD_PRELOAD_32=$LD_PRELOAD_32:/usr/cluster/lib/libschost.so.1
# LD_PRELOAD_64=$LD_PRELOAD_64:/usr/cluster/lib/64/libschost.so.1
# SC_LHOSTNAME=myhost
# export SC_LHOSTNAME LD_PRELOAD_32 LD_PRELOAD_64
# uname -n
myhost
```

Tuning and Troubleshooting

If you are running your applications on a Solaris Cluster system, then service availability is one of your main concerns. The two major causes of service migra-

tion are cluster-node failures and the resource probes detecting that an application is unhealthy.

The Solaris Cluster framework is responsible for detecting and reacting quickly to node failures. When all the cluster private interconnects fail, the default heartbeat settings allow only 10 seconds to elapse before a cluster reconfiguration is triggered and any affected service is restarted on one of the remaining cluster nodes. The application fault probe is responsible for detecting and reacting to an application that is either unhealthy or has failed. Each fault probe can be tuned to take the appropriate action based on the number of failures within a given time interval. Similarly, the resource type `Start` and `Stop` methods for the application are responsible for ensuring that the application is successfully started and stopped. The `Start`, `Stop`, and `Monitor_start` (fault probe) methods have a tunable timeout interval, which is the maximum duration for each operation. The fault probe also has a probe interval, which determines how often the probe is run.

When tuning the probe interval and any associated timeout, you must trade off the impact of the probe on system load and application performance. If you set the values too low, you risk false failover. If you set them too high, you risk waiting longer than necessary to detect a critical problem. Because all applications and workloads are different, the only realistic way to achieve an optimal setting is by thorough testing. You start with high values and gradually reduce them until lower values result in misdiagnosed problems. You then increase the values again to give the fault probe some scope of variability. Of course, all your testing must be performed under realistic workload conditions.

For the start and stop timeouts, your task is slightly easier because the system logs (in `/var/adm/messages`) state how much of the available timeout was used by each method to complete the relevant action. Thus, you can tune your start and stop timeouts such that the application takes no more than, for example, 50 percent of this value under normal conditions.

If it is not obvious why a resource is not functioning correctly, then you can often obtain additional debugging information to help diagnose the problem. The `/var/adm/messages` file is your main source of help. All your resources will log messages here. You can increase the amount of debugging information logged by ensuring that your `syslog.conf` file directs `daemon.debug` output to a suitable file.

Some resource types use flags for specifying the amount of debugging output they produce. Others written in DSDL code (see the section "Data Service Development Library") rely on you specifying a log level from 1 to 9 (with 9 as the maximum) in a file named `/var/cluster/rgm/rt/`*`resource-type-name`*`/loglevel`.

This page intentionally left blank

Index

B

C

D

G

H

I

M

N

S

T

Z