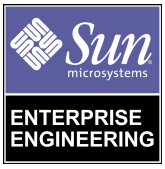
Veritas VxVM Storage Management Software

By Gene Trantham - Enterprise Engineering

Sun BluePrints™ OnLine - May 2000



http://www.sun.com/blueprints

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

Part No.: 806-5596-10
Revision 01, May 2000

Copyright 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X / Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, The Network Is The Computer, Solstice DiskSuite, Jumpstart, Sun BluePrints and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6 / 87) and FAR 52.227-19(6 / 87), or DFAR 252.227-7015(b)(6 / 95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2000 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X / Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, The Network Is The Computer, Solstice DiskSuite, Jumpstart, Sun BluePrints, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Please
Recycle

Adobe PostScript

Veritas VxVM Storage Management Software

Abstract

Most mission critical systems can benefit from storage management software which can mirror, and thereby protect their data against hardware failure. A common storage management tool used on Sun™ systems is produced by Veritas Software Corporation and is sold under the names VxVM and SEVM.

VxVM has a reputation among some system administrators for its complexity in managing storage, and boot devices. The boot disk, and operating system data is perhaps the most critical data that is required to be kept available on a host system. This data is often neglected or improperly protected because of the perceived difficulty involved in the administration process.

This paper explains the underlying actions of VxVM during boot disk encapsulation, and details the mechanism by which it seizes and manages a boot device. A number of best practices will be presented to assist system administrators standardize installations. The direction this paper is focused on availability, serviceability, and administration of data on a boot disk.

Introduction

Most system managers agree that backing up the server data, and even going to the lengths of providing off-site media storage, is a necessary requirement. While this may preserve the data, it does not preserve the availability of the data. A failed disk can be easily replaced and reloaded with the original data from a back up, however, what about lost processing time during repair and data recovery? Also, there is the issue of staleness of the backup and what changes have been made to the dataset between the time of the last backup and the time of the hardware failure.

These issues, among others, led to the development of online data protection software using various forms of RAID. One of the most popular and effective means of preventing an outage due to disk failure is mirroring (RAID_1). However, all mirrors are not created equal. Just providing two copies of the data is not always enough. In fact, it's almost never enough. An understanding of the hardware components necessary to access the underlying disk drive for each copy of the data is required, and also, an understanding of the possible interaction between the various components.

Two of the most popular disk management systems available for the Solaris™ Operating Environment are: Solstice DiskSuite™ software (SDS), and Volume Manager (VxVM).

VxVM, is frequently used on high end systems, but has the reputation of being difficult to configure when dealing with boot devices.

Difficulties can arise during recovery operations, repairs, or even upgrading to a newer version of VxVM. However, this needn't be so. The key to a virtually pain-free and smoothly operating VxVM system, especially with the boot device, is to adhere to some guiding principles that need to be effected at the time of install. These principles can be derived from an understanding of the underlying workings of VxVM, especially in the way it manages a boot disk.

Disk Management Overview

Storage management software achieves transparent mirroring of data with the use of virtual devices. This is basically an extra layer of abstraction between the disk media and the processes above it which need to access and use the devices. These devices are:

Application	vi
FileSystem	ufs device driver
Virtual Device	vxio device driver
Disk Device	sd or ssd device driver

In the case of VxVM, the virtual device is called a volume, with its device driver is vxio. Volumes are presented to the operating system for use as if they were disks. These virtual disks are then available for the support of file systems, swap devices, and raw datafiles.

I/O directed toward the volume is captured by the vxio device driver, which then re-issues them to the underlying disk devices according to the RAID policy for the volume being accessed. A mirrored volume, for example, might take all write requests, then re-issue multiple writes to the various disks which support each pane of the mirror (usually only two panes, but more may exist).

As VxVM is interposed between the devices and the upper layers, it has a measure of control over those disks. However, its authority over the disks is not absolute. Disks are still accessible via their normal device drivers using the conventional logical path name:

```
/dev/dsk/c0t0d0s2
```

For example, a volume supported by a disk at address c0t0d0 does not preclude the use of that disk from other processes. Any other process with sufficient file system permissions may access this disk directly.

All volume managers maintain their own internal database of how the virtual devices should handle I/Os. The configuration database keeps track of which volumes are mirrors, which are RAID-5, what stripe interleave to use, and similar details. This metadata must be stored completely independent from the virtual devices themselves. Both SDS and VxVM store their metadata in a separate private region of each managed disk. The format in which this data is stored is specific to each tool.

The sanctity of the VxVM private regions should not be violated under any circumstances. The volume manager can become confused if the private region of any of its disks undergoes unexpected changes.

At startup, the volume manager will read the configuration information contained in the private regions of any discovered disks. That data is assembled into the configuration database file located inside the vxio device driver. At this point, the volumes are activated and made available to the system for I/Os. If the configuration portion of this process cannot be completed, due to a corrupted private region, for example, one or more volumes may be inaccessible.

Data Protection For All Volumes

A dataset can be protected from certain kinds of hardware errors by storing the data on a mirrored volume. However, the protection is not absolute. A basic definition of mirroring states that there needs to be two or more identical copies of the data. However, there is no requirement that the two copies exist on separate disk devices. A volume in which both copies of the dataset are on the same physical spindle is an example which illustrates that *mirrored* does not always mean protected.

When configuring mirrored volumes with any storage manager, care must be taken to ensure complete device independence for copies of the data. Device independence refers to the individual disk devices, but it can also extend to the series of devices necessary to support the entire data path to the disk. For example, if the two disks supporting a mirror are on the same SCSI bus, the mirror is not device independent, even though the data is on independent disks. The failure of the common path element (the bus itself) will prevent either side of the mirror from being available.

In addition to the bus transport components which might include the host adapter card, SCSI cables and connectors, terminators and the like, the bus also includes a number of unrelated devices which could seriously impact the ability of the bus to carry data.

Suppose an unrelated SCSI device (say, a tape drive) on the same SCSI bus suffered a hardware error which held the bus in reset. This would effectively fail the device path to both the mirror disks even though there are no failures in the pathing elements leading to them, or in the disks themselves.

The solution to this problem is not to move the tape drive. A practical solution would be to move the disk holding a copy of the mirrored dataset to a separate SCSI bus. In this arrangement, if either bus experienced a problem, the other copy of the mirrored data would be unaffected. This approach could be taken for any access element for each copy of the data.

Solaris Operating Environment provides a mechanism to determine the device path components of a disk (or, for any device). The `/devices` directory tree is a way to determine hardware elements leading to a disk:

```
# cd /dev/dsk
# ls -l c0t0d0s2
lrwxrwxrwx   1 root
root  45 Feb  2 17:12 c0t0d0s2
-> /devices/sbus@54,0/SUNW,socal@0,0/sf@0,0/
ssd@w21000020471cb01a,0:a
```

The convenience handle of `/dev/dsk/c0t0d0s2` (often called the logical device path) is a symbolic link to the physical device path found in `/devices`. This physical path shows each of the device drivers used to access that device. Each subdirectory in the pathname represents a discrete hardware element. The device driver and address is indicated in the form:

```
driver@address
```

This method of representing the hardware path as a directory tree is convenient for determining if two devices share a common pathing element in the device tree. In this arrangement, two devices share a common hardware element if they have a common parent in the `/devices` directory tree. The shared parent directory represents the shared element.

The following two devices share the common parent sbus@54,0.

```
c0t0d0s0 -> /devices/sbus@54,0/SUNW,socal@0,0/sf@0,0/
ssd@w21000020471cb01a,0:a
c1t0d0s0 -> /devices/sbus@54,0/SUNW,socal@1,0/sf@0,0/
ssd@w21000020370b0c1b,0:a
```

If these devices were used in support of the two copies in a mirrored dataset, the mirror is said to *depend* on `sbus@54,0`.

Errors or failures on this hardware element have the potential to affect both panes of the mirrored volume. The mean time between failure (MTBF) for this volume is now a function of its weakest link for this one hardware component.

It should be noted that as hardware is added to a system, the mean time between component failures will be reduced. However, failures will not necessarily cause the data to become inaccessible. In essence, the addition of hardware will reduce the time between hardware *component* failures. If a mirrored volume is correctly configured, a component failure will not fail the entire volume. So long as the entire mirror does not depend on the failed component, one or more copies of the data will still be available via other independent components. See Figure 1 - “Shared SCSI Bus vs Split SCSI Bus”

For example, assume that the MTBF for the SCSI card in Figure 1 is 100,000 hours. Configuration A will experience card failure at an average rate of one for every 100,000 hours of operation. Since the mirror is supported by Disk A and Disk B and depends on those components, an outage will result from such a failure. Conversely, Configuration B will suffer a failure of one of its two SCSI cards at an average rate of only once every 50,000 hours of operation. However, this component failure is not sufficient to bring down the volume as the mirror does not depend on both cards.

So far, we have considered the case where only one copy of the data resides on a single disk, with its mirror being on some other single piece of media. However, in practice, mirrors are not limited in this way. Each copy of the data may in fact be another RAID element such as a stripe or concatenation. A common arrangement for VxVM volumes is to mirror two stripes. Each stripe is a RAID_0 object. The pairing of two such objects in a mirror is a RAID_1 configuration. This combination is often referred to as RAID_0+1 (stripe plus mirror).

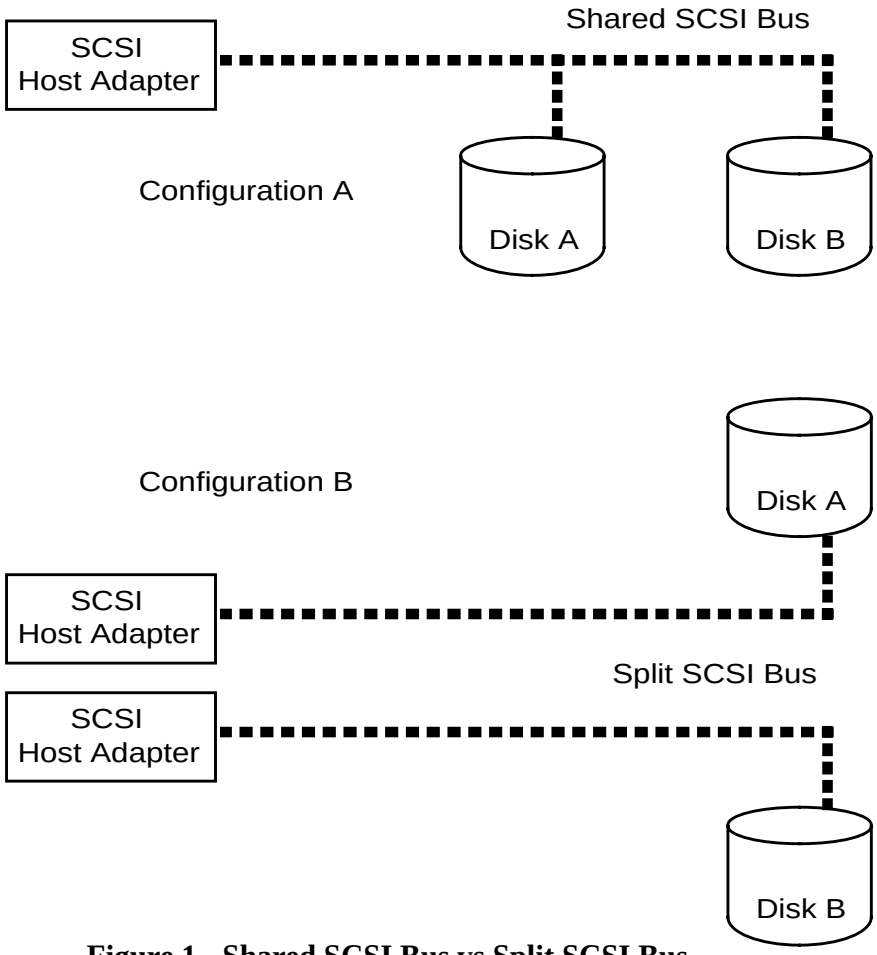


Figure 1 - Shared SCSI Bus vs Split SCSI Bus

This complicates the device independence policy somewhat. If a single copy of the data depends on multiple disks (and their corresponding device paths), there is a greater possibility for overlap with other copies of the data.

Consider the relatively simple mirror illustrated in Figure 2 - “Sample Striped Mirror”

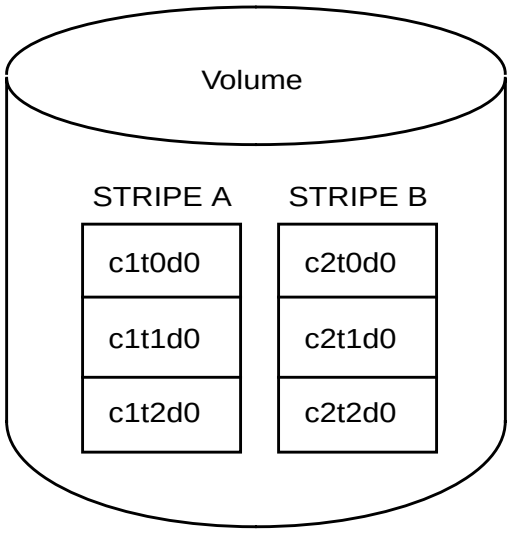


Figure 2 - Sample Striped Mirror (2 x 3 column stripes)

Each element of a stripe (RAID_0) is critical to the integrity of the address space mapped by the stripe. If any one of these disks should fail, the entire stripe is considered unreliable. Therefore, any possible failure that would adversely affect any one of the disks in STRIPE_A does not affect any of the disks in STRIPE_B.

This device path comparison has illustrated that simple mirrors must be expanded to account for all possible pairings of disks between stripes. In the sample configuration, there are 9 such pairings. Any hardware dependency between these will cause the entire volume to *depend* on the common elements found between those two disks.

As the stripes become wider, the number of combinations to be tested grows rapidly. In general, C^n combinations are possible (where C represents columns in each stripe, and n represents the number of panes in a mirror). For example, a mirror consisting of two 5 column stripes, contains 25 potential pairings of disk devices which must be device independent. Testing all these combinations is a time consuming task. Fortunately, much of the analysis can be automated. Appendix A demonstrates a tool which can automate the audit process for RAID1 and RAID0+1 volumes. See Appendix A for sample output from `volchk.pl`.

Boot Disk

Mirrored boot volumes have the potential to be easier or harder to work with than general volumes. The boot volume (referred to as `rootvol`) must not be a stripe or RAID_5 device. It should be composed of simple, one-disk copies. This simplifies the checks necessary to determine device independence. However, boot devices have other special requirements, due to the way the operating system deals with this key device.

Private Regions: The Key To It All

When VxVM assumes ownership of a device, the process is termed initialization. This process involves fencing off a private region on the disk, thereby reducing the space available for data. Private regions typically occupy the first cylinder of the disk, however, this is not a requirement. The remaining cylinders are then available in the *public region* of the disk for the creation of volumes.

Initialization is generally not an issue on most disks under management. Encapsulation, on the other hand, can be difficult. The process of encapsulating a disk is to preserve data already present, while creating one or more cylinders for the private region.

The boot device can present special problems to VxVM if it attempts encapsulation. This is because VxVM is not accustomed to dealing with encapsulation when seizing management control. For example:

If Block 0 is already in use (the boot block and root file system typically begin in cylinder 0), the private regions cannot install there without moving the root file system.

If data is already on the device (which in some cases can consume the entire disk space).

VxVM has some standard workarounds which will handle the majority of circumstances arising from the two example conditions. After a boot disk is encapsulated, remnants of this action may be seen in the `rootdisk-Priv` and `rootdisk-B0` subdisks.

These two subdisks are created in order to preserve two key regions of the disk from being overwritten by volume data.

The two key regions are the VTOC (reserved by `rootdisk-B0`), and the private region (reserved by `rootdisk-Priv`). If a system administrator moved or deleted these two subdisks, their designed protection would be negated.

Formalized management of the private regions is the key to avoiding trouble with the boot device. A best practice is outlined below, and is aimed specifically toward locating the private region on the boot disk in a way that can prevent a multitude of problems which can ensue when dealing with an encapsulated boot device.

VxVM Best Practice For Boot Media

1. Eliminate a separate `/usr` volume

There is little need for `/usr` file system to be separate from the root file system in most environments. Having the `/usr` file system on a volume separate from `rootvol` can complicate service procedures if the primary boot disk needs to be replaced. It would be better to avoid these issues from the outset. All the support utilities for VxVM are located in the `/usr/vxvm` directory. If the system is in the situation where it needs to perform a VxVM operation, yet cannot mount a `/usr` slice or volume, corrective action cannot be taken as a technician will not have access to the tools required. (They are in the inaccessible `/usr` directory)

2. Mirror to a clone disk

The goal of this practice is to duplicate exactly the boot disk to an identical disk. For this to work correctly, the media must be of the same size, performance characteristics, and internal geometry as the original boot device. Ideally, the boot disk and its mirror would be from the same vendor, be the same model number, and be running identical versions of firmware on the internal controller.

Each volume on the boot disk should be supported by identical copies of the data located in identical places on the two disks (the offset address for the start of `rootvol` must be the same on all disks). This will help ensure service and support under emergency repair conditions can be carried out quickly and efficiently.

3. Attach mirrors in geographical order, not alphabetical order.

A common way that administrators mirror a boot device is with the `vxdiskadm` tool. A function within this tool, named "Mirror all volumes on a disk", is a convenient way to do this. However, the `vxdiskadm` function generates a list of the volumes to be mirrored in alphabetical order. Although this achieves the basic goal of mirroring the data, the mirror does not place the data in the optimum locations.

In a typical boot disk encapsulation, the system volumes are: `rootvol`, `swapvol`, `var`, and `opt` (and are generally written in this order on the disk). The `vxdiskadm` function will not mirror these four volumes in the same order in which they appear on the boot disk. First it will attach the volume `opt`, followed by `swapvol`, `rootvol`, and `var`, which is undesirable.

4. Convert the rootdisk from an encapsulated to an initialized disk.

The use of an encapsulated device for the boot disk makes that disk a *special case*. It could be the only encapsulated device in an entire system. This one exception should be removed in order to simplify administration and service procedures. This step is necessary to help ensure the boot disk and its mirror are exact clones. If one is encapsulated and one is initialized, the private regions of these disks will have different addresses. This means that the offsets to the beginning of each volume will be different. This should be avoided.

In addition, by replacing the encapsulated boot disk with an initialized boot disk, the remnants of the encapsulation of `rootdisk-B0`, and `rootdisk-Priv` will be removed from the configuration as they are no longer required.

An effective means of replacing an encapsulated disk with an initialized one is to fail the encapsulated disk and replace it with itself. This procedure will initialize (instead of encapsulating) the replacement drive. Commonly, a typical method by which a disk is failed, and then replaced, is accomplished by using the `vxdiskadm` function. However, as previously discussed, this method does not always reattach the mirrors in an optimum order. An alternate method of replacing an encapsulated disk with an initialized one without using the `vxdiskadm` function is discussed in Appendix A or B.

5. Map volumes to disk slices / partitions for core operating system file systems.

The major file systems necessary to support the fundamentals of an operating system are `root`, `/var`, and `/opt`. If the `/usr` file is on a separate volume, it would also be included in this list. In times of extreme system distress, access to these file systems, regardless of whether VxVM is functioning, is desirable. To ensure this is possible, disks must be re-partitioned to match the volume definitions.

The VxVM tool comes with a utility named `vxmksdpart`, which maps a volume's primary subdisk to a low-level disk slice with the same geometry offsets and length. With the underlying slice available for mount, a technician can use that handle to access the data in an emergency. Even if VxVM is not available, the `vxmksdpart` utility can be found in `/etc/vx/bin` on any standard VxVM installation. Run the command with no arguments to get its usage and help information.

6. Increase the number of configuration file copies to "ALL"

To help prevent loss of the configuration database file, `configdb,` each disk within the disk group should be forced to carry an active copy of the file. The VxVM tool places a backup copy of the file within the private regions of many of its managed disks (but not all). The Volume Manager attempts to distribute this data for resiliency, however, it does not force a copy to each disk. If each group within the disk group carried an active copy of the database configuration file, it would ensure its recovery, even if only one disk survived a major system malfunction.

7. PROM device aliases

Ensure each mirror of the `rootvol` file is bootable. Also, that a device alias exists at the open boot prom (OBP). This will help ensure ease of booting from all such mirrors. The `vxeeprom` function is a way to do this, however, the user-defined aliases can also be edited in the following way:

```
eeprom nvramrc > file

vi file

eeprom "nvramrc=‘cat file‘"
```

8. Provide a fail-safe boot device with VxVM installed.

The Solaris Operating Environment installation CD-ROM is a boot device that can be used when all else fails. Unfortunately, it does not have a copy of the VxVM software. Therefore, it is not possible to boot from the Solaris Operating Environment CD-ROM and still operate VxVM. This poses a problem in situations when the system must boot from the CD-ROM to effect a repair, but cannot use the root volume. There are workarounds for this problem, however, they can be tedious and error prone, except in the most experienced of hands.

An effective method for helping ensure the ability to operate VxVM when all else fails, is to provide a fail-safe boot device which has been prepared for VxVM use. The following lists two ways to do this:

- Reserve a disk within the system as a snapshot disk. Set aside a conventional disk which is not under VxVM control. The disk should be updated with system changes as they occur to the operating system. This snapshot disk will have the VxVM drivers and utilities, but will not depend upon VxVM to boot.
- Add VxVM to the JumpStart™ server via MRtools. The JumpStart server provides a CD-ROM image to any client which attempts to boot from it. This image can be modified to operate VxVM (and other add-on products). Any client which boots from the modified image will then have access to the products. In either case, it is essential to have a boot device not under VxVM control, and yet which loads the VxVM utilities and drivers. This permits the manipulation and repair of VxVM devices without having to compromise the configuration just to get the system to boot.

Appendix A:

Sample output from `volchk.pl`

VxVM Volume Healthcheck for "vol01"

vol01's plexes are listed in the column and row headings of the table below. The hardware in support of each plex is included in the row headings. Each cell of this table lists the hardware found to be in common between the plexes**.**

Table 1:

	vol01-01	vol01-02
vol01-01 iommu0, pln0, sbus0, soc0, ssd25, ssd26	n/a	sbus0, iommu0
vol01-02 iommu0, pln1, sbus0, soc1, ssd31, ssd35	sbus0, iommu0	n/a

```
# /etc/vx/bin/vxdisksetup -i c1t1d1
# vxdg -g rootdg adddisk rootmirror=c1t1d1
# /etc/vx/bin/vxrootmir rootmirror
# vxassist -g rootdg mirror swapvol rootmirror
# vxassist -g rootdg mirror var rootmirror
# vxassist -g rootdg mirror opt rootmirror
# vxdisk -g rootdg list
```

Device Type Disk Group Status
c0t0d0s2 sliced rootdisk rootdg online
c1t1d1s2 sliced rootmirror rootdg online

Note – The rest of this exercise assumes the following bindings:

---

rootdisk=c0t0d0 and rootmirror=c1t1d1. If your device names differ, be sure to substitute accordingly.

```
# vxplex dis rootvol-01
# vxplex dis swapvol-01
# vxplex dis var-01
# vxplex dis opt-01
# vxedit -fr rm rootvol-01 swapvol-01 var-01 opt-01
# vxedit rm rootdiskPriv
```

Appendix B:

Note – `rootdiskPriv` may or may not exist, depending upon how the disk was encapsulated. Remove it if it does exist. Ignore if it does not.

```
# vxdg -g rootdg rmdisk rootdisk
# /etc/vx/bin/vxdisksetup -i c0t0d0
# vxdg -g rootdg adddisk rootdisk=c0t0d0
# /etc/vx/bin/vxrootmir rootvol rootdisk
# vxassist -g rootdg mirror swapvol
# vxassist -g rootdg mirror var
# vxassist -g rootdg mirror opt
```

General Procedure

1) Attach mirrors to `rootvol`, `swapvol`, `var`, `opt` (`vxrootmir`, '`vxassist mirror`')

2) Detach original mirror copies & remove them from configuration ('`vxplex dis`' and '`vxedit rm`')

3) Re-initialize rootdisk. ('`vxdg rmdisk`' and '`vxdisksetup`')

4) Re-attach mirrors to initialized rootdisk (`vxrootmir, 'vxassist mirror'`)

Author's Bio: Gene Trantham

Gene Trantham is a Staff Engineer for Enterprise Engineering at Sun Microsystems. Prior to joining Sun, he spent eight years as a UNIX system administrator specializing in storage management and disaster recovery. While at Sun, Gene has spent most of his time in the field, servicing storage and high-end servers at some of Sun's largest accounts.