CHAPTER 11: TROUBLESHOOTING

by Dr. Albrecht Scriba

11.1 Introduction

Fixing VxVM related problems should never be a game of trial and error. If you are going to mess around in the depths of volume management you need to know what you are doing! For this reason we have decided to write this chapter as a training lesson more than a direct hands-on guide so that you do your training long before you actually need the skill. If you are not familiar with the procedures you should not start repairing serial split brain conditions or other advanced errors in large-scale production systems. So take your time and learn the troubleshooting techniques thoroughly first. There is enough material in this chapter to make you an expert if you care to read it and actually do the exercises. So sit down in front of your VxVM host, get yourself your favorite non-alcoholic beverage and let's begin:

In order to test several troubleshooting scenarios, we need procedures to simulate some kinds of disk outage, because we do not want to actually destroy disks – even if it may take all of your self control not to destroy all those infernal machines with an axe! First, we need to simulate a completely destroyed disk, then a disk temporarily unavailable, but still physically healthy, and finally a disk with some unusable regions on it.

There are several ways to tell VxVM, that a disk is unusable unexpectedly from VxVM's point of view. But note that we probably cannot power off the disk as long as we use disk LUNs provided by a storage array. Other ways have in common, that some relevant data structures are removed from the private region by OS provided tools which, of course, do not inform VxVM, that the disk is regularly unreachable.

Starting with VxVM 5.0, removing the VxVM partitions from a disk does not make the

DOI: 10.1007/978-3-540-85023-6_11, © Springer-Verlag Berlin Heidelberg 2009

disk unusable for VxVM as long as VxVM still has usable disk data within the kernel. To tell VxVM to update or rebuild its kernel database, you must deport and import the corresponding disk group (which is not a realistic scenario) or restart or disable and re-enable **vxconfigd** (which breaks the communication to **vxnotify**). Nevertheless, we will choose the latter procedure, although it is not completely compatible with an actual disk failure.

Simulating a disk temporarily unreachable is performed by the usual steps alluded above and presented below to make a disk unavailable, followed by the reconstruction of the erased private region data. Unfortunately, we do not know a technique to simulate a disk only partially damaged (only some regions on it are not accessible anymore). To achieve this scenario, we would need a kernel layer concatenating real disks or array LUNs to a "disk" visible to the usual disk drivers, and we would make one of those basic disks or LUNs unavailable. Any suggestions are welcome!

To make a disk unusable for VxVM, use the following sequence of commands (you should write a script out of them). We will store the private region data to be removed in order to restore them later, if we want to simulate a temporary disk outage.

Sliced Disk

Get the slice number of the private region, store the first block of 128 kB of the private region data at an appropriate location, overwrite that disk space with zeros, and finally update the VxVM kernel database:

```
# Slice=$(prtvtoc -h /dev/rdsk/c#t#d#s2 | awk '$2==15 {print $1}')
# dd if=/dev/rdsk/c#t#d#s$Slice of=/var/tmp/c#t#d#s$Slice bs=128k count=1
# dd if=/dev/zero of=/dev/rdsk/c#t#d#s$Slice bs=128k count=1
# vxconfigd -k
```

CDS Disk

Nearly the same procedure, except for the different location of the private region (disk offset of 128 kB):

```
# dd if=/dev/rdsk/c#t#d#s2 of=/var/tmp/c#t#d#s2 bs=128k iseek=1 count=1
# dd if=/dev/zero of=/dev/rdsk/c#t#d#s2 bs=128k oseek=1 count=1
# vxconfigd -k
```

If you want to keep the PID of **vxconfigd** (for whatever reason), you may replace the last command for both disk layouts by:

```
# vxdctl disable
# vxdctl enable
```

In order to get a disk we "destroyed" by this procedure usable again for VxVM, we only need to write the stored private region data back to the appropriate slice.

Sliced Disk

Get the slice number of the private region (if you forgot it), and restore the first 128 kB of the private region:

```
# Slice=$(prtvtoc -h /dev/rdsk/c#t#d#s2 | awk '$2==15 {print $1}')
# dd if=/var/tmp/c#t#d#s$Slice of=/dev/rdsk/c#t#d#s$Slice bs=128k
```

CDS Disk
Restore the stored private region data to the appropriate disk offset:

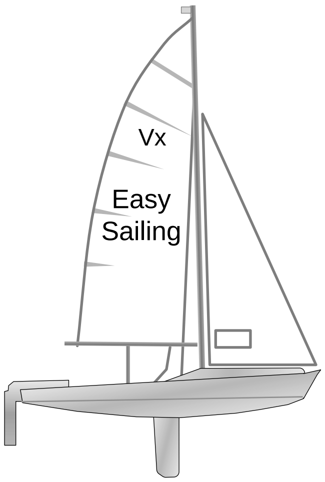
```
# dd if=/var/tmp/c#t#d#s2 of=/dev/rdsk/c#t#d#s2 bs=128k oseek=1
```

During the boot process several VxVM daemons are started, mostly by the script **/etc/rc2.d/S95vxvm-recover** (Solaris 9) and **/lib/svc/method/vxvm-recover** (Solaris 10) respectively. Two of those daemons are of special interest in case of a disk becoming unexpectedly unavailable (for whatever reason): **vxrelocd** (default configuration) or **vxsparecheck**. Both daemons (they must not run at the same time) perform automated tasks to find subdisk or disk replacements for the damaged objects. We will discuss thoroughly their behavior in another part of this chapter. To get an understanding of what happens, if disks encounter problems, we should stop them to prevent VxVM from any automated configuration change.

```
# pkill -9 -xu0 'vxrelocd|vxsparecheck'
```

Having terminated your VxVM troubleshooting tests, you should start the appropriate daemon by issuing one of the following commands:

```
# nohup /usr/lib/vxvm/bin/vxrelocd root &
# nohup /usr/lib/vxvm/bin/vxsparecheck root &
```



11.2 Disk Outage

To test the following scenarios (disk persistently damaged or temporarily inaccessible), please create a disk group containing at least two disks and two volumes in it, each of size 1 GB, one volume (called **vol1** here) without redundancy, another volume (**vol2**) with two plexes/mirrors:

```
# vxdisksetup -i c4t1d0
# vxdisksetup -i c4t2d0
# vxdg init adg adg01=c4t1d0 adg02=c4t2d0
# vxassist -g adg make vol2 1g nmirror=2 init=active alloc=adg01,adg02
# vxassist -g adg make vol1 1g alloc=adg01
```

Issuing I/O to a volume (partially) built on a failed disk will, as a first reaction of the system, show some console messages comparable to the following output example. The disk driver gives up after some retries, then the VxVM Dynamic Multipathing driver (**vxdmp**) disables all its paths to the disk driver (the example only shows one path) and successively, since not a single path but the underlying disk has failed, its DMP node to the disk. Finally, the plex I/O error is reported and the plex "detached" from the volume (still a volume member, but in kernel state DISABLED) due to the failed subdisk belonging to it.

```
Sep 21 18:54:06 sols scsi: WARNING: /pci@1f,0/pci@1/scsi@2/sd@1,0 (sd47):
Sep 21 18:54:06 sols    SCSI transport failed: reason 'incomplete': retrying
command
Sep 21 18:54:08 sols scsi: WARNING: /pci@1f,0/pci@1/scsi@2/sd@1,0 (sd47):
Sep 21 18:54:08 sols    disk not responding to selection
[…]
Sep 21 18:54:25 sols vxdmp: NOTICE: VxVM vxdmp V-5-0-112 disabled path 32/0x178
belonging to the dmpnode 210/0x30
Sep 21 18:54:25 sols vxdmp: NOTICE: VxVM vxdmp V-5-0-111 disabled dmpnode
```

```
210/0x30
Sep 21 18:54:25 sols vxio: WARNING: VxVM vxio V-5-0-151 error on Plex vol2-01
while writing volume vol2 offset 0 length 1
Sep 21 18:54:25 sols vxio: WARNING: VxVM vxio V-5-0-4 Plex vol2-01 detached from
volume vol2
Sep 21 18:54:25 sols vxio: WARNING: VxVM vxio V-5-0-386 adg01-01 Subdisk failed
in plex vol2-01 in vol vol2
```

As long as not strictly instructed to analyze private region access, VxVM keeps the former physical disk state (I/O failure does not automatically imply disk failure). A device scan or a reboot will change the device state to **error** (or **online invalid**, if you only simulate the disk outage).

```
# vxdisk list
[…]
c4t1d0s2     auto:cdsdisk    -            -            online
c4t2d0s2     auto:cdsdisk    adg02        adg          online
[…]
-            -          adg01        adg          failed was:c4t1d0s2
# vxdisk scandisks
# vxdisk list
[…]
c4t1d0s2     auto            -            -            error
c4t2d0s2     auto:cdsdisk    adg02        adg          online
[…]
-            -          adg01        adg          failed was:c4t1d0s2
```

What happens to the VxVM object states, if a disk becomes unavailable, but there are still other healthy disks within the disk group? Of course, the disk group configuration is not lost, because it is stored within multiple private regions (5 as default and if the disk group contains at least 5 disks). VxVM will immediately synchronize the disk group configuration to another disk with a previously inactive configuration, if the damaged disk had an active one. Therefore, the disk group membership of the damaged disk regarding its disk media name, its previous device access, its disk ID, and some other attributes are still known. On the other side, the disk having lost its device access looses its physical membership of the disk group too — in the words of the VxVM documentation: The disk is "detached" from the disk group, as shown by the command output above or the **vxprint** output below:

```
# vxprint -dtg adg
DM NAME         DEVICE       TYPE     PRIVLEN  PUBLEN   STATE

dm adg01        -            -        -        -        NODEVICE
dm adg02        c4t2d0s2     auto     2048     35365968 -
```

All subdisks located on the disk which has become unavailable (state NODEVICE), become unavailable as well (mode NODEVICE, abbreviated NDEV in the output of **vxprint**).

```
# vxprint -stg adg
SD NAME         PLEX         DISK     DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE
[…]
sd adg01-01     vol2-01      adg01    0        2097152  0         -        NDEV
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -        NDEV
sd adg02-01     vol2-02      adg02    0        2097152  0         c4t2d0   ENA
```

What is the effect of subdisks having no underlying physical substrate anymore to the virtual objects of volumes? The answer is quite simple and was already given by the console output: Any plex containing a damaged subdisk is completely cut off from any I/O (application or kernel initiated), even though there may be still other healthy subdisks in it. The application state of the plex enters the NODEVICE state, the kernel state becomes DISABLED, thus preventing the plex from any I/O. If this plex was the last healthy or the only plex within the volume, that is, the volume now has no working instance of its address space, the volume itself enters kernel state DISABLED (no I/O on the volume anymore) by keeping the previous application state (normally ACTIVE).

```
# vxprint -rtg adg
[…]
v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -        NDEV

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol2-01      adg01    0        2097152  0         c4t2d0   ENA
pl vol2-02      vol2         DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         -        NDEV
```

Having physically replaced the damaged disk or chosen another unused disk instead, there are several tools provided by VxVM to repair disk, disk group, subdisk, plex, and volume states. To our experience, best known by most VxVM administrators is the ASCII based interactive tool **vxdiskadm** (especially menus 4 "Remove a disk for replacement" and 5 "Replace a failed or removed disk"). But in many cases, it means breaking a fly on the wheel. Even for the mouse pushers and button pressers having the patience and the memory and processor resources to start the **vea** GUI there are menu items to repair VxVM object states. The command line interface provides some troubleshooting commands, sometimes with obviously overlapping capabilities, thus producing still more trouble in your head. Therefore, we strongly recommend the following list of low-level commands. They are easy to understand, they are straight-forward, they perform only the necessary steps, and they are capable to solve all kind of disk outage with only one small difference at the beginning.

11.2.1 Disk Permanently damaged

Let's start with a completely and irreversibly damaged disk. In case you want to simulate that, look at the procedure and commands mentioned above. Do not forget to stop **vxrelocd** or **vxsparecheck** at first! Be sure to "destroy" the disk which is common to both volumes! Check the results by looking at the output of the **vxdisk** and **vxprint** commands.

1. The damaged disk must be replaced as the first step (in our virtual disk trouble-shooting scenario, we do not need to do anything).

2. Since VxVM does not send repeated test I/Os on a failed disk, nothing happens to the disk state for VxVM. In order to get the disk out of the currently mistaken **error** state, we initiate a device scan. People usually issue the command:

```
# vxdctl enable
```

VxVM will have a look at all disks visible to the OS. In large enterprise environments, this could take several minutes. However, the main purpose of this command is to bring **vxconfigd** in the fully enabled mode during the boot process, thus importing all disk groups except for the already imported boot disk group. To specify only a subset of disks visible to the OS you should use in our example:

```
# vxdisk scandisks device=c4t1d0
```

As a result, the disk becomes **online invalid**, i.e. the disk is physically visible by its VxVM device driver as a healthy disk, but there is no valid private region on it.

```
# vxdisk list
[…]
c4t1d0s2     auto:none       -          -          online invalid
c4t2d0s2     auto:cdsdisk    adg02      adg        online
[…]
-            -         adg01      adg        failed was:c4t1d0s2
```

3. The new disk must be initialized in the same manner as you would initialize any other disk for use by VxVM. The disk becomes completely **online** for VxVM, but is, of course, still not a disk group member.

```
# vxdisksetup -i c4t1d0 […]
# vxdisk list
[…]
c4t1d0s2     auto:cdsdisk    -          -          online
c4t2d0s2     auto:cdsdisk    adg02      adg        online
[…]
-            -         adg01      adg        failed was:c4t1d0s2
```

4. The next step is a little bit tricky to understand. We cannot add the disk to the disk group in the usual manner, because the disk media name (**adg01** in our example) together with its associated attributes already exists within the disk group configuration:

```
# vxdg -g adg adddisk adg01=c4t1d0
VxVM vxdg ERROR V-5-1-559 Disk adg01: Name is already used
```

Choosing another disk media name does not help:

```
# vxdg -g adg adddisk adg03=c4t1d0
# vxdisk list
[…]
c4t1d0s2     auto:cdsdisk    adg03        adg          online
c4t2d0s2     auto:cdsdisk    adg02        adg          online
[…]
-            -         adg01        adg          failed was:c4t1d0s2
```

Now we have a disk group with three configured disk members, one of them still failed and detached from the disk group. Undo this stuff:

```
# vxdg -g adg rmdisk adg03
```

We need a procedure to add a new device access (which may be the same as the previous) to an already known disk group member by keeping (option **-k**) the configured disk attributes:

```
# vxdg -g adg -k adddisk adg01=c4t1d0
# vxdisk list
[…]
c4t1d0s2     auto:cdsdisk    adg01        adg          online
c4t2d0s2     auto:cdsdisk    adg02        adg          online
[…]
```

Finally we got it! The disk group is completely recovered. By adding the new disk to the disk group, all subdisks in NDEV state immediately become ENA (which stands for ENABLED). Since we do not have damaged subdisks within the plexes anymore, their application states NODEVICE are changed to RECOVER (IOFAIL if simulated): They have a physically healthy substrate, but during the disk outage they lost I/O and do not contain the current volume data. Therefore they remain in kernel state DISABLED.

```
# vxprint -rtg adg
[…]
v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED RECOVER  2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -        ENA

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
```

```
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg01-01     vol2-01      adg01    0        2097152  0         c4t2d0  ENA
pl vol2-02      vol2         DISABLED RECOVER  2097152  CONCAT    -       RW
sd adg02-01     vol2-02      adg02    0        2097152  0         -       ENA
```

5. Redundant volumes may have had a working plex during all the time of the disk outage (**vol2** in our example). Plexes in DISABLED kernel state now can be resynchronized:

```
# vxrecover -g adg [-b] [vol2]
# vxprint -rtg adg
[…]
v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -       fsgen
pl vol1-01      vol1         DISABLED RECOVER  2097152  CONCAT    -       RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -       ENA

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -       fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg01-01     vol2-01      adg01    0        2097152  0         c4t2d0  ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg02-01     vol2-02      adg02    0        2097152  0         -       ENA
```

Skipping the volume names as parameters to **vxrecover** will synchronize all volumes within the disk group affected by the damaged disk, but still in kernel state ENABLED. The background option **-b** works as usual: Having started the kernel I/O threads to synchronize, the command returns. Please remember: The volume carries on to be usable by the application in a transactional manner during synchronization (here: atomic copy).

6. Volumes without a healthy plex, thus in kernel state DISABLED (in our example **vol1**) cannot be recovered using **vxrecover**, because they do not have current or valid plex data. For the same reason, they cannot be started regularly. Try it:

```
# vxvol -g adg start vol1
VxVM vxvol ERROR V-5-1-1198 Volume vol1 has no CLEAN or non-volatile ACTIVE
plexes
```

Otherwise, a normal reboot would start such volumes with their applications, even though the volumes do not provide any application data (we replaced the damaged disk). We must forcibly start those volumes, so we can restore application data from backups:

```
# vxvol -g adg -f start vol1
# vxprint -rtg adg vol1
[…]
v  vol1         -            ENABLED  ACTIVE   2097152  SELECT    -       fsgen
pl vol1-01      vol1         ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -       ENA
```

Just to provide an overview, we repeat the steps to recover from complete disk outage without extensive commentaries and misleading commands:

1. Replace the damaged disk.
2. Instruct VxVM to scan the disk access:
   ```
   # vxdisk scandisks device=c#t#d#
   ```
3. Disk recovery:
   ```
   # vxdisksetup -i c#t#d# […]
   ```
4. Disk group and subdisk recovery:
   ```
   # vxdg -g <diskgroup> -k adddisk <failed-dmname>=c#t#d#
   ```
5. Recovery of ENABLED volumes:
   ```
   # vxrecover -g <diskgroup> [-b] [<volumes>]
   ```
6. Start of DISABLED volumes:
   ```
   # vxvol -g <diskgroup> -f start <volumes>
   ```

11.2.2 Disk Temporarily Unavailable

Having successfully repaired a damaged disk and the VxVM state changes, let's turn to a temporary disk outage, e.g. due to temporary power outage of a disk or a disk array (simulated by restoring the private region data).

Now the big surprise! To recover from temporary disk outage, we just need to modify step 1 and skip step 3:

1. Instead of replacing the disk, we must make the disk available again, e.g. by powering on the disk or the disk array. As already mentioned, VxVM does not monitor disk availability (it only monitors path availability according to your DMP configuration), so nothing happens to the VxVM disk and disk group states.

2. Initiate a test for device access to inform VxVM that the disk has come back.

```
# vxdisk scandisks device=c4t1d0
```

Now, initializing the disk by **vxdisksetup** (step 3 above) would be a bad idea. It would overwrite a still valid private region, possibly with a different size, thus maybe overwriting application data on the first subdisk or at least placing the subdisks at different physical disk offsets and, in consequence, spoiling still existing application data. Skip this step, by all means!

Steps 3 to 5 are completely identical to steps 4 to 6 of the former procedure. Unlike a damaged disk, a disk temporarily unavailable will provide the best possible application data when forcibly starting a disabled volume.

To sum up once again, but for temporary disk outage:

1. Make the disk available again.
2. Instruct VxVM to scan the disk access:

```
    # vxdisk scandisks device=c#t#d#
```

3. Disk group and subdisk recovery:

```
    # vxdg -g <diskgroup> -k adddisk <failed-dmname>=c#t#d#
```

4. Recovery of ENABLED volumes:

```
    # vxrecover -g <diskgroup> [-b] [<volumes>]
```

5. Start of DISABLED volumes:

```
    # vxvol -g <diskgroup> -f start <volumes>
```

11.2.3 Replacing an OS Disk

We still need to turn toward special problems when replacing a damaged OS disk. We assume that you have mirrored your OS volumes, so you still have a running system.

Why is it not sufficient to execute the already described procedure to replace a failed disk, if the new disk is intended to hold a mirror of the OS volumes? Well, during the first stage of the boot process, the Open Boot Prom reads the disk VTOC to identify the boot device partition carrying tag 2 and containing the root file system to be mounted later on /. The kernel root device (this is not necessarily identical to the boot device) may not be based upon a partition driver. But even in case your OS will be accessed by VxVM volumes, the root device **/pseudo/vxio@0:0** specified in **/etc/system** needs a partition for the root file system to build a kernel RAM configured volume on it.

Therefore, if you are trying to replace a failed OS disk using the regular procedure, you will encounter at least one, probably even two problems. First, you cannot boot from the replaced disk because it has no partitions on it. Remember, mirroring volumes on a disk does not mean mirroring a disk's physical structure. So at least, we need to put appropriate partitions over the subdisks created by volume mirroring. Secondly, since partitions are always aligned to cylinder boundaries, offset and length of the original subdisks probably do not fit to different disk hardware.

To avoid any problems like that, it is useful to remove subdisks configured on the failed disk together with their associated plexes and to recreate the mirrors according to the cylinder specifications of the new disk. We do not spoil any redundancy because, due to the failed disk, we already lost volume redundancy. We only remove VxVM objects currently not having a physical disk substrate. The following is the commented procedure when replacing a failed OS disk:

1. Replace the damaged disk by a healthy one or select another free disk. Be sure to use a disk providing the required space to mirror all volumes of the still working OS disk.

2. Have VxVM scan the new device.

```
# vxdisk scandisks device=c#t#d#
```

3. Initialize the disk for VxVM use, that is, create a sliced disk layout which is required for OS disks. If necessary, limit the size of the private region to 1 MB, for example (VxVM 5.0 is built to create 32 MB as default).

```
# vxdisksetup -i c#t#d# format=sliced [privlen=1m]
```

4. Add to the still known (we assume **osdg02**), but physically failed disk group member a new (and possibly different) device access:

```
# vxdg -g bootdg -k adddisk osdg02=c#t#d#
```

5. So far so good, no big difference to the standard procedure! But now, remove all subdisks configured on the failed OS disk together with their associated plexes. Given the standard output of **vxprint**, this is hard work, even harder if scripted. But the early developers of VxVM were obviously passionate Unix guys. They designed **vxprint** as a powerful tool! An approach step by step:

List all plex names of the boot disk group:

```
# vxprint -g bootdg -pn
```

List all subdisks configured on disk **osdg02**:

```
# vxprint -g bootdg -se 'sd_dmname="osdg02"'
```

First combination: List all plex names having at least one associated subdisk on disk **osdg02**:

```
# vxprint -g bootdg -pne 'pl_sd.sd_dmname="osdg02"'
```

Now the final wizardry using Shell command substitution (poor who still does remove the plexes command by command):

```
# vxplex -g bootdg -o rm dis $(
  vxprint -g bootdg -pne 'pl_sd.sd_dmname="osdg02"')
```

6. Instead of volume recovery we add a mirror to the OS volumes being still active on the other OS disk (we assume **osdg01**). The VxVM script **vxmirror** we execute takes care, that the new subdisks are aligned to disk cylinder boundaries, and builds appropriate partitions over these subdisks (for a detailed procedure with explanations see the "Technical Deep Dive" part). Fortunately, **vxmirror** prints out a log of issued commands to support our understanding of the procedure.

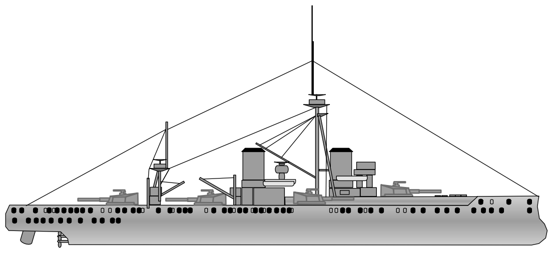
```
# vxmirror -g bootdg osdg01 osdg02
! vxassist -g bootdg mirror rootvol osdg02
! vxassist -g bootdg mirror swapvol osdg02
! vxassist -g bootdg mirror var osdg02
! vxassist -g bootdg mirror opt osdg02
! vxbootsetup -g bootdg osdg02
```

The script **vxmirror**, as shown by the listed commands, mirrors all volumes built of subdisks on **osdg01** which are currently not redundant. The second disk specification **osdg02** is optional and denotes the disk to use for the mirrors — mostly unnecessary because we mostly have only two disks within the boot disk group. Furthermore, by invoking **vxbootsetup**, it builds partitions on the new subdisks, writes a boot block on the root partition (obviously as a fallback, because the volume mirroring as a raw device procedure already did copy it), and creates, if missing, an alias entry in the OBP **nvramrc** (but not in **boot-device**).

Maybe you remember: This step is completely identical to the procedure to add a mirror to an encapsulated OS disk explained in chapter 10. The "Full Battleship" part will add some remarks on creating a partition based **vfstab** as a boot process troubleshooting prerequisite.

The summary of these steps:

1. Replace the failed OS disk.
2. Instruct VxVM to scan the disk access:
   ```
   # vxdisk scandisks device=c#t#d#
   ```
3. Disk recovery:
   ```
   # vxdisksetup -i c#t#d# format=sliced […]
   ```
4. Disk group and subdisk recovery:
   ```
   # vxdg -g bootdg -k adddisk <failed-dmname>=c#t#d#
   ```
5. Remove all failed subdisks and their associated plexes:
   ```
   # vxplex -g bootdg -o rm dis $(
     vxprint -g bootdg -pne 'pl_sd.sd_dmname="<failed-dmname>"')
   ```
6. Add mirrors to the volumes and place partitions over the subdisks:
   ```
   # vxmirror -g bootdg <active-dmname> [<failed-dmname>]
   ```


The Full Battleship

11.3 Disk Outage in Detail

11.3.1 A Complete Disk Array Temporarily Unavailable

Failure of a single disk, whether totally or temporarily, is a scenario rarely occurring in today's data centers, except maybe of an OS disk. Most data centers rely on intelligent disk arrays providing a configurable set of virtual LUNs based on the known RAID concepts. If just one physical disk of the array fails, the disk array administrator is notified to replace the failed disk, while the LUNs used by host applications remain fully accessible due to the RAID based redundancy.

A more widespread failure is the temporary outage of a complete disk array, e.g. by power failure of the disk array or of a site of a data center, or by loss of all paths to that array. Bringing back the accessibility of the array to its client hosts leads maybe to a terribly high number of LUNs detached from their disk groups (we assume the members of the disk group spread over more than one array, so the disk group is not completely unavailable; for the latter scenario see page 363). Of course, step number 3 explained above to recover these disks detached from their disk groups (**`vxdg -k adddisk …`**) will work properly, but that would mean an awful lot of commands to execute.

Well, fortunately, on Unix systems we have a powerful Shell. So it is quite easy to issue a loop on all failed disks currently detached from their disk group, but at the same time VxVM initialized by recovering their device access. To prevent unexpected results, we make sure, that those disks are really identical to the already configured disk group members and not freshly initialized disks with different physical properties. A simple way to check the identity may be achieved by comparing the disk IDs. As explained on page 91 of the Disk Group chapter each time a disk is initialized by VxVM, it is tagged by a unique disk ID comprising the initialization date in Unix time format, an internal counter, and the initializing VxVM host ID, separated by dots.

The following command lists the current disk ID associated with the physical device access:

```
# vxdisk -s list c4t1d0
[…]
diskid: 1213183912.30.sols
```

[…]

To show the disk ID associated with the disk media name as part of the disk group configuration, we use the **vxprint** command. Note that this command does not require physical disk access. Since the VxVM kernel module **vxspec** is the source of imported disk group information, we may execute it even against disks physically failed.

```
# vxprint -g adg -F %diskid adg01
1213183912.30.sols
```

To demonstrate the necessary steps to recover multiple disks, we provide a simple code fragment to recover all those disks currently detached due to a temporary failure by skipping freshly initialized disks:

```
# List all disks
vxdisk -q list |
# Filter detached disks, print DMName, DGroup, last DAName
nawk '$1=="-" {sub(/^was:/,"",$NF); print $3,$4,$NF}' |
while read DMName DGroup DAName; do
    # Get current disk ID
    CurrDiskID=$(vxdisk -s list $DAName | nawk '$1=="diskid:" {print $2; exit}')
    # Get disk ID configured in disk group
    DGConfDiskID=$(vxprint -g $DGroup -F %diskid $DMName)
    # Not identical? Next disk
    [[ $CurrDiskID != $DGConfDiskID ]] && continue
    # Add device access
    vxdg -g $DGroup-k adddisk $DMName=$DAName
done
```

Since this task should happen more often than a single disk failure, VxVM provides a script called **vxreattach** to recover a set of temporary failed disks. Its basic strategy looks like the code fragment above, but it is, of course, surrounded by a lot of check commands, not to forget the extensive copyright notes. Its usage is:

```
# vxreattach [-r] [-b] <daname> [<daname> ...]
```

Adding the option **-r** also recovers all plexes within enabled volumes, which were disabled (according to step 4 above). As usual, **-b** ("background") terminates the **vxreattach** command after having started the kernel threads to synchronize the stale plexes.

11.3.2 A Disk Group Temporarily Inaccessible

Our previous topic assumed that the disk group configuration is still not lost. What happens if all your disks holding the active disk group configuration become unavailable at once? You might consider this a scenario rarely occurring. But sometimes your disk group is built from disks of just one disk array. Are you ready to exclude a power failure of the disk array?

Even if you neatly spread your disks over more than one array, you cannot deny the possible risk to loose your disk group configuration. If one array fails temporarily, VxVM automatically switches all its active disk group configuration disk roles to the remaining array. If the first mentioned disk array comes back, all active configuration disks are kept at the latter array! Now that array suffers from a power failure or, even worse, gets damaged...

The current section deals with a temporary outage of the disk group configuration disks (persistent loss will be explained later). No physical substrate for storing VxVM objects, for logging object states, and for modifying attribute values is available. Volumes using these disks will get I/O failures on their subdisks, thus VxVM disables the affected plexes and volumes, if all of their plexes are disabled. You will see a lot of error messages on the console sent from the device drivers and the **vxdmp** and **vxio** kernel modules (compare page 352). Instead of our agitated expressions above, we may put oil on troubled waters given this scenario: As long as VxVM keeps disk group configuration data in the kernel, the disk group remains online in spite of some or all of its disks being inaccessible. There is, to calm you down a little bit more, an easy way to store the disk group configuration held by the kernel memory to still healthy, but currently inactive private regions. Simply make all disks of the disk group to active configuration and log disks:

```
# vxedit -g adg set nconfig=all adg
# vxedit -g adg set nlog=all adg
# vxdg list adg
[…]
copies:    nconfig=all nlog=all
config:    seqno=0.1457 permlen=1280 free=1273 templen=4 loglen=192
config disk c4t1d0s2 copy 1 len=1280 state=clean online
config disk c4t2d0s2 copy 1 len=1280 state=clean online
config disk c4t3d0s2 copy 1 len=1280 state=clean online
[…]
log disk c4t1d0s2 copy 1 len=192
log disk c4t2d0s2 copy 1 len=192
log disk c4t3d0s2 copy 1 len=192
[…]
```

Marking disks as active configuration disks will flush to them the disk group configuration stored in the kernel. If you discard your kernel disk group configuration, e.g. by restarting **vxconfigd** in order to re-read the non-existent physical disk group configuration, before flushing it to other disks, you will encounter problems:

```
# vxconfigd -k
VxVM vxconfigd ERROR V-5-1-583 Disk group adg: Reimport of disk group failed:
        Disk group has no valid configuration copies
# vxdisk -g adg list
VxVM vxdisk ERROR V-5-1-607 Diskgroup adg not found
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
[…]
c4t1d0s2     auto:cdsdisk    adg01        adg          online dgdisabled
```

```
c4t2d0s2     auto:cdsdisk    adg02         adg          online dgdisabled
[…]
```

There is only one way to recover a **dgdisabled** disk group: Reattach the disks, so that at least one valid disk group configuration copy is accessible, and re-import (maybe forced) the disk group (which requires application stop, if not already stopped).

```
# vxdg deport adg
# vxdg [-f] import adg
# vxdisk -g adg list
DEVICE       TYPE            DISK          GROUP        STATUS
c4t1d0s2     auto:cdsdisk    adg01         adg          online
c4t2d0s2     auto:cdsdisk    adg02         adg          online
# vxrecover -g adg -s [-b]
```

11.3.3 A Partially Failed Disk ("Failing")

A scenario indeed rarely occurring is a partial disk failure, that is the disk is still visible, but some regions on it are damaged. Given virtual LUNs this should not happen. A disk whose public region is still accessible by VxVM, but nevertheless encountered I/O errors due to unreadable subdisks, is not detached from the disk group, but remains a disk group member. That disk is only marked as **failing** by VxVM (as shown in the output of **vxdisk** and **vxprint**):

```
# vxdisk -g adg list
DEVICE       TYPE            DISK          GROUP        STATUS
c4t1d0s2     auto:cdsdisk    adg01         adg          online failing
c4t2d0s2     auto:cdsdisk    adg02         adg          online
# vxprint -dtg adg
DM NAME         DEVICE       TYPE     PRIVLEN  PUBLEN   STATE

dm adg01        c4t1d0s2     auto     2048     35365968 FAILING
dm adg02        c4t2d0s2     auto     2048     35365968 -
```

The **failing** flag set by VxVM prohibits allocation of free space of this disk when creating new subdisks during volume creation, when searching for subdisk replacement by hot relocation, and so on.

If a disk gets partially damaged, VxVM sets the **failing** flag on it and marks affected subdisks as FAIL (**vxprint -t** output, column MODE). But this is not a one-to-one condition. If you see the **failing** flag on a disk, you cannot conclude that the disk must be partially damaged. In most cases, the cable connection to the disk is somewhat instable, that is, you can issue I/O in general, but sometimes an I/O request fails. Therefore, the first step is to determine, if the disk or the cabling to the disk is damaged.

If you determine, that you had cable problems, and you can fix it, you should remove the misleading **failing** flag (this is never done automatically by VxVM):

```
# vxedit -g adg set failing=off adg01
```

In case of a partially damaged disk, you should replace the disk as soon as possible. Maybe there are still other usable subdisks on that disk, whose data you want to move to another disk within the disk group. An easy way to do so is to execute:

```
# vxevac -g <diskgroup> <dmname-to-evac> [<dmname-as-replacement> ...]
```

vxevac is a Shell script invoking **vxassist move** commands on volumes with subdisks on the specified disk. Unfortunately, the script is designed to exit the evacuation process if a volume move fails. In our case of a failing disk, this will happen for the damaged subdisk. So you must move the other volumes manually:

```
# vxassist -g <diskgroup> move <volume> alloc=\!<dmname-to-evac>,\
  [<dmname-as-replacement>,...]
```

In either case, the damaged subdisk will remain on the failing disk. How can you replace that disk? Of course, you could simply perform a physical removal of the disk, thus producing a completely failed disk within VxVM. This would send e-mails, trigger SNMP consoles, and so on. That's why we should use a procedure telling VxVM that we indeed do want to remove the disk. This is a little bit tricky:

```
# vxdg -g adg rmdisk adg01
VxVM vxdg ERROR V-5-1-552 Disk adg01 is used by one or more subdisks.
        Use -k to remove device assignment.
```

Of course, we cannot remove a disk group member still in use by subdisks, even if these subdisks are physically unusable. But they are still known to the disk group configuration. We need to keep (**-k**) the disk group membership and the configured disk attributes by "detaching" the disk from the disk group, that is, to remove only the device assignment:

```
# vxdg -g adg -k rmdisk adg01
```

This disk now looks like a failed disk except for the object states: REMOVED instead of NODEVICE. Note that the **failing** flag is cleared by this procedure:

```
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
[…]
c4t1d0s2     auto:cdsdisk    -            -            online
c4t2d0s2     auto:cdsdisk    adg02        adg          online
[…]
-            -          adg01        adg          removed was:c4t1d0s2
# vxprint -dtg adg
DM NAME         DEVICE       TYPE     PRIVLEN  PUBLEN   STATE

dm adg01        -            -        -        -        REMOVED
dm adg02        c4t2d0s2     auto     2048     35365968 -
# vxprint -rtg adg
[…]
v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED REMOVED  2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -        RMOV

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         DISABLED REMOVED  2097152  CONCAT    -        RW
sd adg01-01     vol2-01      adg01    0        2097152  0         -        RMOV
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c4t2d0   ENA
```

The steps to replace the removed disk correspond exactly to the steps to replace a failed disk, as described and explained on page 352 above. But maybe, in addition, you will want to move back the subdisks to their original disk media location afterwards:

```
# vxassist -g <diskgroup> move <volume> \!<dmname-as-replacement> <orig-dmname>
```

11.3.4 Hot Relocation

VxVM provides two mutually exclusive mechanisms handling disk failures by automatically searching for disk or subdisks replacement: **vxrelocd** or **vxsparecheck**. Our previous tests simulating disk outage were unrealistic to some extent, for we stopped the running instance of this process pair in order to understand the basic principles of disk recovery. It is time we demonstrate and explain how these mechanisms work. The default process starting with VxVM 3.0 is **vxrelocd**, so this is our first candidate.

The script **vxrelocd**, started into background by **/etc/rc2.d/S95vxvm-recover** (Solaris 9) or **/lib/svc/method/vxvm-recover** (Solaris 10), itself invokes **vxnotify** as a child process by capturing and evaluating STDOUT of the latter process for disk and plex detach messages (**-f**) awaiting 15 sec. (**-w 15**) after the last detach event before responding to the failure.

```
# ptree -a $(pgrep -xu0 vxrelocd)
1     /sbin/init -s
  2786  /sbin/sh - /usr/lib/vxvm/bin/vxrelocd root
    3747  /sbin/sh - /usr/lib/vxvm/bin/vxrelocd root
      3748  vxnotify -f -w 15
```

The following commands together with their output simulate a failed disk, demonstrate the behavior of hot relocation and explain the steps necessary to recreate the original configuration.

1. We create a mirrored (**vol2**) and a non-redundant volume (**vol1**), both having in common a subdisk on disk **adg01**.

```
# vxassist -g adg make vol2 1g layout=mirror nmirror=2 init=active \
  alloc=adg01,adg02
# vxassist -g adg make vol1 1g alloc=adg01
# vxprint -rtg adg
[…]
dm adg01        c1t1d0s2     auto     2048     122171136 -
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 -

v  vol1         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         c1t1d0   ENA

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol2-01      adg01    0        2097152  0         c1t1d0   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

2. The disk failure is simulated by overwriting the private region and restarting **vxconfigd**. Some seconds later, we notice an atomic copy synchronization thread to resynchronize the plex of **vol2** affected by the disk failure (for details see the "Technical Deep Dive" section).

```
# dd if=/dev/zero of=/dev/rdsk/c1t1d0s2 bs=128k oseek=1 count=1
# vxconfigd -k
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
 50555 50555     ATCOPY/R 28.61% 0/2097152/600064 PLXATT vol2 vol2-01 adg
```

3. As proved by the output of **vxprint**, **vol1** has lost its single and last healthy plex, no automatic recovery is possible, the volume is DISABLED. During resynchronization of **vol2**, we already recognize that the failed plex **vol2-01** got a new subdisk **adg03-01** on another disk (**adg03**). After synchronization has completed, the plex is a full member of the volume, entering application state ENABLED and mode RW. Hot relocation has fulfilled its primary task.

Note that in case of a more complicated plex layout (concatenating and/or striping of subdisks) the whole affected plex becomes disabled due to the failed subdisk and must be synchronized completely. Replacement on a per-subdisk level by hot relocation does not imply resynchronization just on a per-subdisk base. Layering the volume subdisk by subdisk (if possible) or at least column by column would help to reduce synchronization I/O. On the other side, adding a DCL volume with its capability to track volume region changes does not avoid full plex synchronization — strange enough!

```
# vxprint -rtg adg
[…]
dm adg01        -            -        -        -        NODEVICE
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 -

v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -        NDEV

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  STALE    2097152  CONCAT    -        WO
sd adg03-01     vol2-01      adg03    0        2097152  0         c1t1d2   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
# vxprint -rtg adg
[…]
dm adg01        -            -        -        -        NODEVICE
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 -

v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -        NDEV

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg03-01     vol2-01      adg03    0        2097152  0         c1t1d2   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

4. **vxrelocd** was invoked with the parameter **root**, specifying a mail recipient, meaning **root@localhost**. You should extend the list of mail recipients according to your needs by editing **/etc/rc2.d/S95vxvm-recover** (Solaris 9) or **/lib/svc/method/vxvm-recover** (Solaris 10). During the process of hot relocation, several mails are sent (sorted in chronological order):

```
Subject: Volume Manager failures on host haensel
[…]
Failures have been detected by the VERITAS Volume Manager:

failed disks:
 adg01

failed plexes:
vol1-01
vol2-01

The Volume Manager will attempt to find spare disks, relocate failed
subdisks and then recover the data in the failed plexes.
```

```
Subject: Volume Manager failures on host haensel
[…]
Unable to relocate failed subdisk from plex vol1-01 because no
suitable mirror was found from which to recover data.

Subject: Volume Manager failures on host haensel
[…]
Attempting to relocate subdisk adg01-01 from plex vol2-01.
Dev_offset - length 2097152 dm_name adg01 da_name c1t1d0s2.
The available plex vol2-02 will be used recover the data.

Subject: Attempting VxVM relocation on host haensel
[…]
VERITAS Volume Manager is preparing to relocate for diskgroup adg.
Saving the current configuration in:
/etc/vx/saveconfig.d/adg.080928_095541.mpvsh

Subject: Attempting VxVM relocation on host haensel
[…]
Volume vol2 Subdisk adg01-01 relocated to adg03-01, but not yet recovered.

Subject: Volume Manager failures on host haensel
[…]
Recovery complete for volume vol2 in disk group adg.
```

5. Our final goal is still not reached. **vol1** is further on unusable due to its DISABLED kernel state. The failed disk **adg01** should be reattached to the disk group. Although **vol2** has regained its redundancy by implementing a previously free disk, we should move the subdisk currently named **adg03-01** back to its original location in order to completely recover the former configuration. Since the latter task needs some further explanation, we separate it (step 6) from the current section.

```
# vxdisksetup -i c1t1d0
# vxdg -g adg -k adddisk adg01=c1t1d0
# vxvol -g adg -f start vol1
# vxprint –rtg adg
[…]
dm adg01        c1t1d0s2     auto     2048     122171136 -
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 -

v  vol1         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         c1t1d0   ENA

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
```

```
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg03-01     vol2-01      adg03    0        2097152  0         c1t1d2   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

6. Our final task: placing the "hot relocated" subdisk to its original disk. We already know several ways with specific advantages and disadvantages:

```
# vxmake -g adg sd adg01-01 disk=adg01 offset=0 len=1g
# vxsd -g adg -d mv adg03-01 adg01-01
# vxedit -g adg rm adg03-01
```

Advantage: Full control on the position of the subdisk

Disadvantage: Where do we know the original location and name of the subdisk? Furthermore: Some tricky work

Note: The option **-d** of the **vxsd mv** command discards the hot relocation information stored as subdisk attributes, see below.

```
# vxassist -g adg move vol2 alloc=\!adg03,adg01
```

Advantage: Not too much work, because on a per-volume base

Disadvantage: Once again, where do we know the original location and name of the subdisk? No control on the position of the subdisk on the target disk, subdisks of the volume intentionally placed on **adg03** are moved as well

```
# vxevac -g adg adg03 adg01
```

Advantage: Just one simple command

Disadvantage: Once again, where do we know the original location and name of the subdisk? No control on the position of the subdisk on the target disk, all subdisks on **adg03** are moved

Fortunately, a subdisk provides an attribute pair designed to bail us out, i.e. to support subdisk move tasks after hot relocation has taken place:

```
# vxprint -g adg -m adg03-01
[…]
        orig_dmname=adg01
        orig_dmoffset=0
# vxprint -g adg -F '%name %orig_dmname %orig_dmoffset' \
  -e 'sd_orig_dmname="adg01"'
adg03-01 adg01 0
```

The hot relocation process obviously stored the original disk media name and the public region offset of the subdisk on that disk. It did not save as subdisk attribute values the length of the original subdisk (in most cases identical to the current length, see note below) and its name (a VxVM object together with all attribute values holds 256 bytes within the private region, storing the former subdisk name would waste space). But recall a mail sent by **vxrelocd** mentioning a configuration saving file under **/etc/vx/saveconfig.d**, where you can extract the missing information from.

In case of disk group members carrying sliced layout, we may encounter some rounding-up of the length of the subdisk to disk cylinder boundaries. Note that the **vxassist** default layout comprises **diskalign**:

```
# vxassist help showattrs
#Attributes:
 layout=nomirror,nostripe,nomirror-stripe,nostripe-mirror,
nostripe-mirror-col,nostripe-mirror-sd,
noconcat-mirror,nomirror-concat,span,nocontig,raid5log,noregionlog,diskalign,
nostorage
[…]
```

We provide a commented code fragment to relocate subdisks to their original location:

```
# We need current subdisk name, original disk media name and offset, and length
Fields='%name %orig_dmname %orig_dmoffset %len'
# Subdisk counter
((SdCount=0))

# Print subdisk attributes, if generated by hot relocation
vxprint -g adg -se 'sd_orig_dmname!=""' -F "$Fields" |
while read SdName SdDisk SdOffset SdLen; do
    # New subdisk name, based on original disk name + "UR" + counter
    UrSdName=${SdDisk}-UR-$((SdCount+=1))
    # Recreate original subdisk
    vxmake -g adg sd $UrSdName disk=$SdDisk offset=$SdOffset len=$SdLen ||
    continue
    # Move subdisk content and swap plex association, discard hot reloc info
    vxsd -g adg -d mv $SdName $UrSdName || continue
    # Remove hot relocation generated subdisk
    vxedit -g adg rm $SdName
done
```

A somewhat similar procedure (see the "Technical Deep Dive" section) was implemented into the VxVM script **vxunreloc**. While our code fragment relocated all subdisks to their original location, **vxunreloc** just relocates subdisks originating from one disk specified by a required command parameter. While we were bound to the exact subdisk positions, **vxunreloc** is able to round up to disk specific cylinder boundaries (option **-f**). Finally, **vxunreloc** allows for the specification of an alternate target disk (option **-n**).

```
# vxunreloc -g adg adg01
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
```

```
 50568          ATCOPY/R 20.80% 0/2097152/436224 SDMV adg03-01 adg01-UR-001 adg
# vxprint -rtg adg vol2
[…]
v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl %1           vol2         ENABLED  TEMPRM   2097152  CONCAT    -        WO
sd adg01-UR-001 %1           adg01    0        2097152  0         c1t1d0   ENA
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg03-01     vol2-01      adg03    0        2097152  0         c1t1d2   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

After hot relocation has terminated, we could rename the subdisk:

```
# vxprint -rtg adg vol2
[…]
v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-UR-001 vol2-01      adg01    0        2097152  0         c1t1d0   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
# vxedit -g adg rename adg01-UR-001 adg01-01
# vxprint -rtg adg vol2
[…]
v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol2-01      adg01    0        2097152  0         c1t1d0   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

The hot relocation procedure does not need disks specified as spare disks, any space on any regular disk (see below) may serve as "spare parts inventory". But a disk marked as spare disk will have higher priority for the implemented subdisk selection policy **spare=yes** – but only if the volume move requires more than one disk (see "Technical Deep Dive"). If you need to restrict subdisk replacement only to spare disks, enter **spare=only** into **/etc/default/vxassist** (**spare=no** would exclude spare disks from hot relocation).

If you want to exclude a disk from the "spare parts inventory" by keeping the default selection policy **spare=yes**, e.g. in order not to degrade its performance, mark it as **nohotuse**. Obviously, **spare** and **nohotuse** are mutually exclusive. We recall the commands:

```
# vxedit -g adg set spare=on|off adg01
# vxedit -g adg set nohotuse=on|off adg01
```

At first sight, the **vxrelocd** procedure looks "finely granulated", avoiding any unnecessary I/O and being capable to spread multiple subdisks originating from one failed disk to several target disks in case we did not provide a spare disk of appropriate space. But at second thought, what would you like to do with a disk partially unusable? Of course, you would like to evacuate it as soon as possible. Evacuating means finding replacements for

ALL subdisks residing on the affected disk, not just for the failed subdisks. Then, you surely would want to replace it by a new one and finally to relocate all moved subdisks.

11.3.5 Hot Spare

The legacy technique in VxVM 2.x, called "Hot spare", provided a procedure similar to what we described in the last phrase of the antecedent paragraph and may be still reactivated in lieu of hot relocation:

```
# pkill -9 -xu0 vxrelocd
# nohup /usr/lib/vxvm/bin/vxsparecheck root &
# vi /etc/rc2.d/S95vxvm-recover (Solaris 9)
# vi /lib/svc/method/vxvm-recover (Solaris 10)
[…]
# vxrelocd root &          (commented out)
[…]
vxsparecheck root &        (removed comment sign)
[…]
```

Hot spare works on a per-disk base: Each failed disk will be replaced by another disk holding appropriate space to cover all subdisks of the failed disk. Replacing disks must be marked as spare disks. For details, see the following example on our well-known volumes **vol1** and **vol2**:

1. We select a disk of the disk group to become a spare disk. Our tries to create a volume demonstrate, that a spare disk is excluded from the list of generically available disks as long as we do not explicitly allocate it by storage attributes.

```
# vxdisk -g adg list
DEVICE       TYPE            DISK         GROUP        STATUS
c1t1d0s2     auto:cdsdisk    adg01        adg          online
c1t1d1s2     auto:cdsdisk    adg02        adg          online
c1t1d2s2     auto:cdsdisk    adg03        adg          online
# vxedit -g adg set spare=on adg03
# vxdisk -g adg list
DEVICE       TYPE            DISK         GROUP        STATUS
c1t1d0s2     auto:cdsdisk    adg01        adg          online
c1t1d1s2     auto:cdsdisk    adg02        adg          online
c1t1d2s2     auto:cdsdisk    adg03        adg          online spare
# vxassist -g adg make demovol 1g layout=stripe ncol=3
VxVM vxassist ERROR V-5-1-435 Cannot allocate space for 2097152 block volume
# vxassist -g adg make demovol 1g layout=stripe ncol=3 alloc=adg01,adg02,adg03
# vxedit -g adg -rf rm demovol
# vxprint -rtg adg
[…]
dm adg01        c1t1d0s2     auto     2048     122171136 -
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 SPARE

v  vol1         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         c1t1d0   ENA

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol2-01      adg01    0        2097152  0         c1t1d0   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

2. We "destroy" disk **adg01** holding subdisks for both volumes in order to see how the **vxsparecheck** daemon handles a disk failure. The first effect of a failed disk is of course, independent on hot relocation or hot spare working or not, that it is detached from the disk group, that all affected plexes enter kernel state DISABLED, and that volumes loosing their last healthy plex become DISABLED as well. In addition to this basic procedure, **vxsparecheck** will immediately mark subdisks to be placed on the spare disk as RLOC (for "relocate", see the output of an additionally started **vxnotify**).

```
# dd if=/dev/zero of=/dev/rdsk/c1t1d0s2 bs=128k oseek=1 count=8
# vxconfigd -k
# vxnotify -f -w 15
[…]
detach disk c1t1d0s2 dm adg01 dg adg dgid 1223834080.19.haensel
detach plex vol2-01 volume vol2 dg adg dgid 1223834080.19.haensel
relocate subdisk adg01-01 plex vol2-01 volume vol2 dg adg dgid
1223834080.19.haensel
disabled vol vol1 rid 0.1155 dgid 1223834080.19.haensel adg
detach plex vol1-01 volume vol1 dg adg dgid 1223834080.19.haensel
[…]
# vxprint -rtg adg
[…]
dm adg01        -            -        -        -         NODEVICE
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 SPARE

v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         -        NDEV

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg01-01     vol2-01      adg01    0        2097152  0         -        RLOC
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

3. The second action of **vxsparecheck** is, after the default waiting period of 15 seconds, to exclusively reserve the spare disk and to forcibly remove the failed disk (device access **c1t1d0s2**), both visible in command outputs only for few seconds. Then, **vxsparecheck** swaps the names of the spare disk and the removed disk: The disk media name **adg01**, previously belonging to **c1t1d0s2**, gets the new device access **c1t1d2s2**, while **adg03** enters the REMOVED state. The subdisks, previously storing their data on **c1t1d0s2**, are now placed on **c1t1d2s2**. Since these subdisks get a physically healthy disk, the affected plexes change their application state from NODEVICE to RECOVER (IOFAIL, if simulated). The failed disk itself is then completely removed, even as a disk group member.

```
# vxprint -dtg adg
[…]
dm adg01        -            -        -        -         NODEVICE
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 RESERVED
# vxprint -dtg adg
[…]
dm adg01        c1t1d2s2     auto     2048     122171136 -
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        -            -        -        -         REMOVED
# vxprint -rtg adg
[…]
dm adg01        c1t1d2s2     auto     2048     122171136 -
dm adg02        c1t1d1s2     auto     2048     122171136 -

v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED RECOVER  2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         c1t1d2   ENA

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         DISABLED RECOVER  2097152  CONCAT    -        RW
sd adg01-01     vol2-01      adg01    0        2097152  0         c1t1d2   RLOC
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

4. Finally, synchronization threads are started in order to bring the subdisks residing on the new device to the current volume states (if possible). The plex associated to an ongoing ENABLED volume enter into the STALE state and, after resynchronization, into ACTIVE. The non-redundant volume **vol1** remains DISABLED.

```
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
 50575           PARENT/R  0.00% 1/0(1) VXRECOVER adg01 adg
 50576 50576     ATCOPY/R 21.58% 0/2097152/452608 PLXATT vol2 vol2-01 adg
# vxprint -rtg adg
[…]
```

```
dm adg01       c1t1d2s2    auto     2048     122171136 -
dm adg02       c1t1d1s2    auto     2048     122171136 -

v  vol1        -           DISABLED ACTIVE   2097152  SELECT    -       fsgen
pl vol1-01     vol1        DISABLED RECOVER  2097152  CONCAT    -       RW
sd adg01-02    vol1-01     adg01    2097152  2097152  0         c1t1d2  ENA

v  vol2        -           ENABLED  ACTIVE   2097152  SELECT    -       fsgen
pl vol2-01     vol2        ENABLED  STALE    2097152  CONCAT    -       WO
sd adg01-01    vol2-01     adg01    0        2097152  0         c1t1d2  RLOC
pl vol2-02     vol2        ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg02-01    vol2-02     adg02    0        2097152  0         c1t1d1  ENA
# vxprint -rtg adg
[…]
v  vol1        -           DISABLED ACTIVE   2097152  SELECT    -       fsgen
pl vol1-01     vol1        DISABLED RECOVER  2097152  CONCAT    -       RW
sd adg01-02    vol1-01     adg01    2097152  2097152  0         c1t1d2  ENA

v  vol2        -           ENABLED  ACTIVE   2097152  SELECT    -       fsgen
pl vol2-01     vol2        ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg01-01    vol2-01     adg01    0        2097152  0         c1t1d2  ENA
pl vol2-02     vol2        ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg02-01    vol2-02     adg02    0        2097152  0         c1t1d1  ENA
```

5. Like **vxrelocd**, **vxsparecheck** was invoked with the parameter **root** (or with an extended list of mail recipients) and dispatched two mails during our failure and recovery example (sorted in chronological order):

```
Subject: Volume Manager failures on host haensel
[…]
VxVM vxsparecheck NOTICE V-5-2-191 Failures have been detected by the VERITAS
Volume Manager:

failed disks:
 adg01

failed plexes:
 vol1-01
 vol2-01

failed subdisks:



failed volumes:
 vol1
VxVM vxsparecheck ERROR V-5-2-533
The Volume Manager will attempt to find hot-spare disks to replace any
```

```
failed disks and attempt to resyncrhonize the data in the failed plexes
from plexes on other disks.

VxVM vxsparecheck ERROR V-5-2-330
The data in the failed volumes listed above is no longer available. It
will need to be restored from backup.
```

```
Subject: Attempted VxVM recovery on host haensel
[…]
VxVM vxsparecheck INFO V-5-2-279
Replacement of disk adg01 in group adg with disk device
c1t1d2s2 has successfully completed and recovery is under way.


VxVM vxsparecheck ERROR V-5-2-373 The following volumes:

vol1

occupy space on the replaced disk, but have no other enabled mirrors
on other disks from which to perform recovery. These volumes must have
their data restored.
VxVM vxsparecheck INFO V-5-2-434
To restore the contents of any volumes listed above, the volume should
be started with the command:

        vxvol -f start <volume-name>

and the data restored from backup.
```

6. Unlike **vxunreloc** for **vxrelocd**, the final, manual recovery after disk replacement is not supported by a VxVM script. So we "replace" and recover the "failed" disk and restore the original configuration (if necessary): The subdisk replacement in **vol2** is moved back to the previous device access. **vol1**, holding no reasonable application data, is best recovered by forcibly deleting the subdisk and adding a new one located on the new disk (a subdisk move would mean synchronization of unusable data, so why bother?).

```
# vxedit -g adg rename adg01 adg03
# vxdisksetup –i c1t1d0
# vxdg -g adg adddisk adg01=c1t1d0
# vxdisk -g adg list
DEVICE       TYPE            DISK         GROUP        STATUS
c1t1d0s2     auto:cdsdisk    adg01        adg          online
c1t1d1s2     auto:cdsdisk    adg02        adg          online
c1t1d2s2     auto:cdsdisk    adg03        adg          online
# vxassist -g adg -b move vol2 alloc=\!adg03,adg01
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
```

```
 50581          ATCOPY/R 25.78% 0/2097152/540672 SDMV adg01-01 adg01-02 adg
# vxprint -rtg adg vol2
[…]
v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl %2           vol2         ENABLED  TEMPRM   2097152  CONCAT    -        WO
sd adg01-02     %2           adg01    0        2097152  0         c1t1d0   ENA
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol2-01      adg03    0        2097152  0         c1t1d2   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
# vxsd -g adg -f -o rm dis adg01-02
# vxprint -rtg adg vol1
[…]
v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED(SPARSE) RECOVER 0 CONCAT    -        RW
# vxdg -g adg free adg01
DISK         DEVICE       TAG          OFFSET    LENGTH    FLAGS
adg01        c1t1d0s2     c1t1d0       2097152   120073984 -
# vxmake -g adg sd adg01-02 disk=adg01 offset=2097152 len=2097152
# vxsd -g adg assoc vol1-01 adg01-02
# vxprint -rtg adg vol1
[…]
v  vol1         -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         DISABLED RECOVER  2097152  CONCAT    -        RW
sd adg01-02     vol1-01      adg01    2097152  2097152  0         c1t1d0   ENA
# vxvol -g adg -f start vol1
# vxedit -g adg set spare=on adg03
# vxprint -rtg adg
[…]
dm adg01        c1t1d0s2     auto     2048     122171136 -
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 SPARE

v  vol1         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-03     vol1-01      adg01    2097152  2097152  0         c1t1d0   ENA

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-02     vol2-01      adg01    0        2097152  0         c1t1d0   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol2-02      adg02    0        2097152  0         c1t1d1   ENA
```

11.4 Synchronization Tasks

11.4.1 Optimizing Resynchronization

If a disk has failed and was replaced by a new one, the affected plexes do not contain reasonable application data, they must be completely synchronized. If a disk or a disk array is only inaccessible for a certain period and then comes back, those plexes contain stale application data. Depending on the duration of inaccessibility and the amount of application write I/Os, they are still physically synchronized in many or most regions of the volume.

Given a simple mirrored volume, VxVM does not know where the synchronization of regions may be skipped, so a complete synchronization is necessary. It would be a good idea to mark regions affected by write I/Os starting from the time when the plexes became disabled. After bringing online the temporarily failed disks by the procedure explained above, VxVM would need to synchronize only some regions of the plexes with a noticeable reduction of sync I/O.

A data change object (DCO) of version 20 together with its associated data change log volume (DCL) is capable not only to perform fast mirror synchronization of snapshots, not only to provide instant or space optimized snapshots, and not only to serve as a replacement to the former plex based Dirty region log. It also tracks the regions of write I/Os to a mirrored volume, if one or more of its plexes become unavailable. Let's test it!

1. We create three mirrored volumes within a disk group consisting of two disks. The first volume (**vol1**) will keep the nologging layout, **vol2** gets a DCO of version 0, while **vol3** is supplied with a DCO of version 20. Note the different commands to create the different versions of a DCO. Also note the nice wizardry of **vxprint** to generate a formatted output of some specified attributes (option **-F**). Finally note that the DCL volume to DCO version 0 is smaller in size (144 sectors) than the volume to version 20 (544 sectors). The latter is designed to fulfill more tasks than the former (see chapter 9 on snapshots).

```
# vxassist -g adg make vol1 1g nmirror=2 init=active
# vxassist -g adg make vol2 1g nmirror=2 init=active
# vxassist -g adg addlog vol2 logtype=dco
```

The last two commands may be shortened to:

```
# vxassist -g adg make vol2 1g layout=mirror,log nmirror=2 logtype=dco \
  init=active
```

The third volume:

```
# vxassist -g adg make vol3 1g nmirror=2 init=active
# vxsnap -g adg prepare vol3
# vxprint -rLtg adg
```

```
[…]
V  NAME         RVG/VSET/CO  KSTATE   STATE    LENGTH   READPOL   PREFPLEX UTYPE
PL NAME         VOLUME       KSTATE   STATE    LENGTH   LAYOUT    NCOL/WID MODE
SD NAME         PLEX         DISK     DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE
DC NAME         PARENTVOL    LOGVOL
[…]
v  vol1         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol1-01      adg01    0        2097152  0         c4t1d0   ENA
pl vol1-02      vol1         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol1-02      adg02    0        2097152  0         c4t2d0   ENA


v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-02     vol2-01      adg01    2097152  2097152  0         c4t1d0   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-02     vol2-02      adg02    2097152  2097152  0         c4t2d0   ENA
dc vol2_dco     vol2         vol2_dcl

v  vol2_dcl     -            ENABLED  ACTIVE   144      SELECT    -        gen
pl vol2_dcl-01  vol2_dcl     ENABLED  ACTIVE   144      CONCAT    -        RW
sd adg01-04     vol2_dcl-01  adg01    6291456  144      0         c4t1d0   ENA
pl vol2_dcl-02  vol2_dcl     ENABLED  ACTIVE   144      CONCAT    -        RW
sd adg02-04     vol2_dcl-02  adg02    6291456  144      0         c4t2d0   ENA


v  vol3         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol3-01      vol3         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-03     vol3-01      adg01    4194304  2097152  0         c4t1d0   ENA
pl vol3-02      vol3         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-03     vol3-02      adg02    4194304  2097152  0         c4t2d0   ENA
dc vol3_dco     vol3         vol3_dcl

v  vol3_dcl     -            ENABLED  ACTIVE   544      SELECT    -        gen
pl vol3_dcl-01  vol3_dcl     ENABLED  ACTIVE   544      CONCAT    -        RW
sd adg01-05     vol3_dcl-01  adg01    6291600  544      0         c4t1d0   ENA
pl vol3_dcl-02  vol3_dcl     ENABLED  ACTIVE   544      CONCAT    -        RW
sd adg02-05     vol3_dcl-02  adg02    6291600  544      0         c4t2d0   ENA
# F='%{name:-10} %{version:2} %{drl:-4} %{drllogging:-4}'
# vxprint -g adg -F "$F" vol2_dco vol3_dco
vol2_dco    0 off  off
vol3_dco   20 on   on
```

2. We make a disk of the disk group used by all volumes unavailable for VxVM, perform write I/O of 1 MB on each volume, and recover the temporarily failed disk and the disk group.

```
# dd if=/dev/rdsk/c4t1d0s2 of=/var/tmp/c4t1d0s2 bs=128k iseek=1 count=8
# dd if=/dev/zero of=/dev/rdsk/c4t1d0s2 bs=128k oseek=1 count=8
# vxconfigd -k
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
[…]
c4t1d0s2     auto:cdsdisk    -            -            online invalid
c4t2d0s2     auto:cdsdisk    adg02        adg          online
[…]
-            -          adg01        adg          failed was:c4t1d0s2
# for i in 1 2 3; do
  dd if=/dev/zero of=/dev/vx/rdsk/adg/vol$i bs=128k count=8; done
# dd if=/var/tmp/c4t1d0s2 of=/dev/rdsk/c4t1d0s2 bs=128k oseek=1
# vxdisk scandisks device=c4t1d0
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
[…]
c4t1d0s2     auto:cdsdisk    -            -            online
c4t2d0s2     auto:cdsdisk    adg02        adg          online
[…]
-            -          adg01        adg          failed was:c4t1d0s2
# vxdg -g adg -k adddisk adg01=c4t1d0
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
[…]
c4t1d0s2     auto:cdsdisk    adg01        adg          online
c4t2d0s2     auto:cdsdisk    adg02        adg          online
[…]
```

3. Before resynchronizing the volumes, we reset the I/O counters of VxVM for the disk group. We expect five synchronization tasks: three tasks for the three top-level volumes, and two tasks for the two mirrored DCL volumes. After resynchronization, we check for mirror sync I/Os, i.e. for I/O types "Atomic Copy" and "Read-Writeback". Since we had a failed disk producing one disabled plex within each volume, we do not expect Read-Writebacks, but you never know...

```
# vxstat -g adg -r
# vxrecover -bg adg
# vxtask list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   364           PARENT/R  0.00% 5/0(1) VXRECOVER
   365   365     ATCOPY/R 00.98% 0/2097152/20480 PLXATT vol1 vol1-01 adg
# vxtask list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   364           PARENT/R 20.00% 5/1(1) VXRECOVER
   369   369     ATCOPY/R 16.02% 0/2097152/335872 PLXATT vol2 vol2-01 adg
```

```
# vxstat -g adg -f ab
                          ATOMIC COPIES              READ-WRITEBACK
TYP NAME                OPS    BLOCKS AVG(ms)      OPS    BLOCKS AVG(ms)
vol vol1               1024   2097152   71.3         0         0    0.0
vol vol2               1024   2097152   61.8         0         0    0.0
vol vol2_dcl              1       144   20.0         0         0    0.0
vol vol3                  1      2048   80.0         0         0    0.0
vol vol3_dcl              1       544   30.0         0         0    0.0
```

Indeed, the command output exactly provides the information we expected before: five synchronization tasks mentioned in the first output line of the two **vxtask list** command examples after the title. But the most important information lies in the output of **vxstat**: Without mirror logging or with a DCO version 0, the volume was resynchronized completely: 1024 I/O operations at 2048 sectors (= 1 MB) each, totally 1 GB. However, the third volume was synchronized just with 1 I/O at 1 MB (remember, we wrote 1 MB new data on each volume). The DCL volumes, due to their small size not furnished with another DCO structure, are synchronized completely.

11.4.2 Controlling Synchronization Behavior

In most cases, VxVM selects the appropriate synchronization mechanisms to recreate full volume redundancy. If a plex is in a state known to specify stale or inaccessible data (such as NODEVICE, IOFAIL, RECOVER, STALE, OFFLINE), while there is at least one healthy plex (ACTIVE, CLEAN, under certain circumstances EMPTY and SNAPDONE) within the volume, then VxVM uses the "Atomic Copy" procedure, thus reading the volume based on the healthy plexes and writing these data to the bad plex. Application write I/Os are written to all plexes, read requests are just taken from the healthy plexes before or during resynchronization.

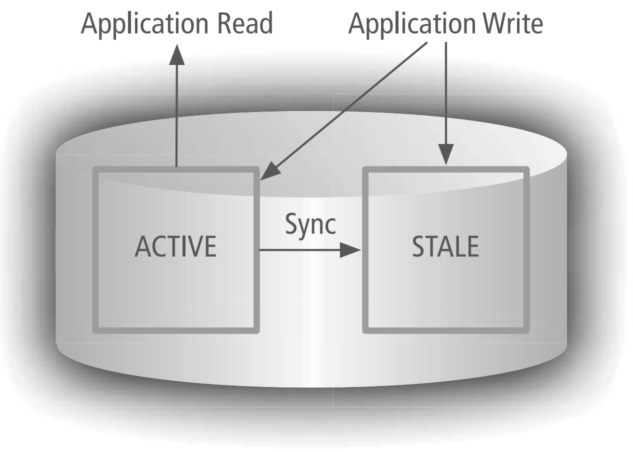


Figure 11-1: "Atomic Copy" Synchronization

If all plexes are in a healthy state, but VxVM knows, that there is need to synchronize (e.g. the **devopen** volume attribute is **on**, because the system crashed, while there was write I/O to the raw device, or the volume with its file system was mounted), VxVM does not spoil the still existing redundancy on nearly all parts of the volume (as some less intelligent volume manager software does by marking one data copy as bad). All healthy plexes remain healthy, for load balancing reasons the synchronization direction changes every 256 MB, and for performance reasons the round robin read policy remains in use. To provide a transactional raw device, VxVM immediately writes back every read I/O to the other plexes to ensure data integrity when reading those data blocks once again, but probably from the other plex. Therefore, this technique is called "Read-Writeback". The topic of Read-Writeback synchronisation is difficult to understand, and you want to refer to the additional coverage in the Volume chapter, page 132.

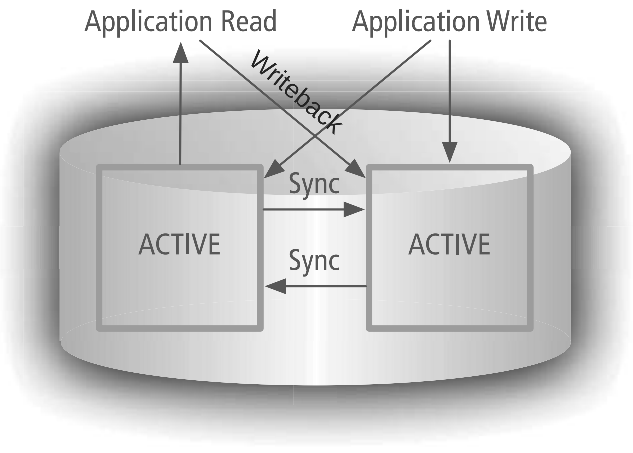


Figure 11-2: "Read-Writeback" Synchronization

If all plexes of a volume are in bad states, VxVM does nothing except for forbidding to start the volume in the regular way. Manual intervention is required.

Even if VxVM could synchronize the volume's plexes due to the existence of a healthy plex, we sometimes do not want the synchronization method VxVM would choose. So we need ways to modify plex states to specify another synchronization procedure or even to completely skip plex synchronization.

The ways to change plex states are somewhat tricky, and the scenarios making such modifications necessary sometimes academic. We will provide some realistic scenarios to show examples for the use of the **vxmend** command and some **vxvol** keywords.

Atomic Copy Redirection: Data Recovery from Offlined Plexes

It is a good idea to freeze volume data in a mirrored volume by offlining a data plex when applying software upgrades or patches, modifying the application configuration, and so on. Upgrades and patches could make things even worse, a misconfigured application cannot be started. A data plex may be offlined at any time, independent from the volume state:

```
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  ACTIVE   2097152  SELECT    -       fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0  ENA
pl vol-02       vol          ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1  ENA
# vxmend -g adg off vol-02
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  ACTIVE   2097152  SELECT    -       fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0  ENA
pl vol-02       vol          DISABLED OFFLINE  2097152  CONCAT    -       RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1  ENA
```

In case the upgraded, patched, or modified data work fine, we soon should synchronize the offlined plex according to the current volume data in order to establish data redundancy.

```
# vxmend -g adg on vol-02
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  ACTIVE   2097152  SELECT    -       fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0  ENA
pl vol-02       vol          DISABLED STALE    2097152  CONCAT    -       RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1  ENA
```

Of course, the plex contains stale data and is therefore still disabled for any I/O. Atomic copy resynchronization is not started automatically, we issue volume recovery.

```
# vxrecover -g adg -b vol
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   251   251      ATCOPY/R 11.62% 0/2097152/243712 PLXATT vol vol-02 adg
```

During synchronization, the synchronized plex is available for application write I/Os (kernel state ENABLED), but blocked for read access (mode WO).

```
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  ACTIVE   2097152  SELECT    -       fsgen
```

```
pl vol-01       vol         ENABLED  ACTIVE    2097152  CONCAT    -        RW
sd adg01-01     vol-01      adg01    0         2097152  0         c1t1d0   ENA
pl vol-02       vol         ENABLED  STALE     2097152  CONCAT    -        WO
sd adg02-01     vol-02      adg02    0         2097152  0         c1t1d1   ENA
```

If we crashed the application due to the "high enterprise quality" of a patch, our offlined plex provides most probably the most up-to-date application data. So we must switch the synchronization direction. It is impossible and indeed of no use to enable access to another data set (the offlined plex), while the volume is in use. This would spoil any transactional device characteristics. Consequently, we need to stop application and volume first. Our example unmounts the file system on the volume, but you may assume any other required application termination in order to free the raw device from I/O.

```
# umount /mnt
# vxvol -g adg stop vol
# vxprint -rtg adg vol
[…]
v  vol          -           DISABLED CLEAN     2097152  SELECT    -        fsgen
pl vol-01       vol         DISABLED CLEAN     2097152  CONCAT    -        RW
sd adg01-01     vol-01      adg01    0         2097152  0         c1t1d0   ENA
pl vol-02       vol         DISABLED OFFLINE   2097152  CONCAT    -        RW
sd adg02-01     vol-02      adg02    0         2097152  0         c1t1d1   ENA
```

The corrupted data set (**vol-01**) is still marked as healthy by VxVM (CLEAN), the correct data set (**vol-02**) is still in OFFLINE state. We want to mark **vol-01** as STALE and **vol-02** as CLEAN or ACTIVE to force the desired synchronization direction.

```
# vxmend -g adg on vol-02
# vxprint -rtg adg vol
[…]
v  vol          -           DISABLED CLEAN     2097152  SELECT    -        fsgen
pl vol-01       vol         DISABLED CLEAN     2097152  CONCAT    -        RW
sd adg01-01     vol-01      adg01    0         2097152  0         c1t1d0   ENA
pl vol-02       vol         DISABLED STALE     2097152  CONCAT    -        RW
sd adg02-01     vol-02      adg02    0         2097152  0         c1t1d1   ENA
# vxmend -g adg fix clean vol-02
VxVM vxmend ERROR V-5-1-1182 Volume vol contains plexes in the CLEAN state
```

VxVM does not like your attack to its knowledge of missing data integrity, because two clean plexes would mean a fully synchronized, healthy volume. The keyword **active** instead of **clean** would work, but then the volume would be marked as NEEDSYNC, thus evoking read-writeback synchronization. Our task needs another procedure.

```
# vxmend -g adg fix stale vol-01
# vxmend -g adg fix clean vol-02
# vxprint -rtg adg vol
[…]
```

```
v  vol          -            DISABLED CLEAN    2097152  SELECT    -        fsgen
pl vol-01       vol          DISABLED STALE    2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          DISABLED CLEAN    2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
```

This works! The roles of good and stale plex are swapped. Now enable and resynchronize the volume in the correct direction. Both commands listed below are in this case absolutely identical. Finally, restart your application.

```
# vxvol -g adg [-o bg] start vol
# vxrecover -g adg [-b] -s vol
# mount -F <fstype> /dev/vx/dsk/adg/vol /mnt
```

Atomic Copy Instead of Read Writeback: One Plex Is Few I/Os Farther than the Other

We assume a total, but cascading power outage in a data center. The volume, mirrored across two LUN arrays, first looses on the physical layer one data plex and some seconds later the other one. Due to the timeouts of the stateless Fibre Channel protocol, both plexes keep their healthy ACTIVE state, but the latter contains more current data. After powering on the data center, VxVM would resynchronize the volume using the read-writeback method. If we cannot recover the volume data generated during the last seconds before the total power outage by application recovery strategies (e.g. the online redo logs of an Oracle database reside on the same LUN arrays), we can tell VxVM, that we do not have two healthy plexes, but just one. Since the boot process already starts resynchronization (**/etc/rc2.d/S95vxvm-recover** Solaris 9, **/lib/svc/method/vxvm-recover** Solaris 10), we must boot into the single user mode or milestone. Remember: **/etc/rcS.d/S35vxvm-startup1** for the OS disk group and **/etc/rcS.d/S85vxvm-startup2** (Solaris 9, same file names in Solaris 10 under **/lib/svc/method**) for other disk groups respectively just start volumes without synchronization, marking them as NEEDSYNC and thus forcing read writeback application I/Os (**vxrecover -s -n**).

```
ok boot -s
[…]
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  NEEDSYNC 2097152  SELECT    -        fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
# vxmend -g adg off vol-02
# vxmend -g adg on vol-02
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  NEEDSYNC 2097152  SELECT    -        fsgen
```

```
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          DISABLED STALE    2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
# exit
```

The continued boot process will now resynchronize the volume using the atomic copy method (**/etc/rc2.d/S95vxvm-recover** or **/lib/svc/method/vxvm-recover**):

```
vxrecover -b -o iosize=64k > /dev/null
```

Force Atomic Copy Synchronization: I/O on Disks Outside of VxVM

As long as all disk I/Os are managed by the VxVM kernel, its integrity states are correct. But VxVM does not hinder the device drivers to accept I/Os issued outside of VxVM. It is quite easy to execute a **dd** command immediately on partition drivers, generating mirror inconsistency, where VxVM believes mirrors to be coherent. As soon as possible, we must mark the damaged plex as a bad plex. Probably you should stop the application to prevent it from decisions based on wrong data. But you are not forced by VxVM to stop the volume.

```
# vxmend -g adg off vol-02
# vxmend -g adg on vol-02
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  NEEDSYNC 2097152  SELECT    -        fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          DISABLED STALE    2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
# vxrecover -g adg [-b] vol
```

One command less, but executable only against a stopped volume:

```
# vxmend -g adg fix stale vol-02
# vxprint -rtg adg vol
[…]
v  vol          -            DISABLED CLEAN    2097152  SELECT    -        fsgen
pl vol-01       vol          DISABLED CLEAN    2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          DISABLED STALE    2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
# vxrecover -g adg [-b] -s vol
```

We will discuss another example in a boot troubleshooting section (booting without VxVM).

Force Read-Writeback Synchronization: Suspicious Plex Data

You fear differences in very small regions of your mirrors, because you forgot to disable disk track caching, and your data center suffered a power outage. Small data sets still not written physically to the disk platters but already committed to the volume layer (the **devopen** attribute is already set to **off**) could be dangerous for your application. Therefore, you decide to synchronize the mirrors, but you do not know, which mirror is good and which is bad (probably they are both affected by data loss). To keep redundancy, you prefer to choose the read-writeback method.

```
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
# vxrecover -g adg vol
```

The recovery command will not resynchronize the volume, because, according to VxVM, the volume is still synchronized. Otherwise, you would see the volume state NEEDSYNC. Since a read-writeback synchronization could change the current volume content (a data block was read from the first plex by the application, but is synchronized from the second plex by VxVM), the volume must be stopped.

```
# vxvol -g adg stop vol
# vxmend -g adg fix empty vol
# vxprint -rtg adg vol
[…]
v  vol          -            DISABLED EMPTY    2097152  SELECT    -        fsgen
pl vol-01       vol          DISABLED EMPTY    2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          DISABLED EMPTY    2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
```

The volume now looks and behaves like a freshly created volume. No plex has priority over the other one, so VxVM will choose read-writeback synchronization.

```
# vxvol -g adg [-o bg] start avol
    or:
# vxrecover -g adg [-b] -s avol
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   176         RDWRBACK/R 69.73% 0/2097152/1462272 VOLSTART vol adg
```

Skip Synchronization: Volume Data Unusable

As already explained in the Volume chapter on page 120, you may create new mirrored volumes without synchronization either by a sequence of **vxmake** commands and finally by starting the DISABLED/EMPTY volume executing:

```
# vxvol -g adg init active vol
```

Or you may issue the **vxassist** command with a special parameter:

```
# vxassist -g adg make vol 1g layout=mirror init=active
```

In case VxVM tells you that the volume must be synchronized, but you are not interested in the volume data, because you want to restore the volume data from a backup image, you could believe it would be sufficient to mark the volume as DISABLED/EMPTY:

```
# vxprint -rtg adg vol
[…]
v  vol          -           ENABLED  NEEDSYNC 2097152  SELECT    -         fsgen
pl vol-01       vol         ENABLED  ACTIVE   2097152  CONCAT    -         RW
sd adg01-01     vol-01      adg01    0        2097152  0         c1t1d0    ENA
pl vol-02       vol         ENABLED  ACTIVE   2097152  CONCAT    -         RW
sd adg02-01     vol-02      adg02    0        2097152  0         c1t1d1    ENA
# vxvol -g adg stop vol
# vxprint -rtg adg vol
[…]
v  vol          -           DISABLED NEEDSYNC 2097152  SELECT    -         fsgen
pl vol-01       vol         DISABLED ACTIVE   2097152  CONCAT    -         RW
sd adg01-01     vol-01      adg01    0        2097152  0         c1t1d0    ENA
pl vol-02       vol         DISABLED ACTIVE   2097152  CONCAT    -         RW
sd adg02-01     vol-02      adg02    0        2097152  0         c1t1d1    ENA
# vxmend -g adg fix empty vol
VxVM vxmend ERROR V-5-1-1211 Volume vol has plexes in the ACTIVE state, use -o
force
# vxmend -g adg -o force fix empty vol
# vxprint -rtg adg vol
[…]
v  vol          -           DISABLED EMPTY    2097152  SELECT    -         fsgen
pl vol-01       vol         DISABLED EMPTY    2097152  CONCAT    -         RW
sd adg01-01     vol-01      adg01    0        2097152  0         c1t1d0    ENA
pl vol-02       vol         DISABLED EMPTY    2097152  CONCAT    -         RW
sd adg02-01     vol-02      adg02    0        2097152  0         c1t1d1    ENA
# vxvol -g adg init active vol
# vxprint -rtg adg vol
[…]
v  vol          -           ENABLED  NEEDSYNC 2097152  SELECT    -         fsgen
pl vol-01       vol         ENABLED  ACTIVE   2097152  CONCAT    -         RW
```

```
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
```

Oops! VxVM did not forget the need for synchronization. The volume looked like a freshly created volume, but only within the standard output of **vxprint**. A volume attribute kept the information for recovery: **cdsrecover_seqno**, set to a value not equal to 0 (mostly 1). Unfortunately, this attribute value cannot be reset to 0. We only know one way to dupe VxVM: Store the volume attributes recursively into a file, edit the value of **cdsrecover_seqno**, rebuild the volume and initialize it by skipping synchronization:

```
# vxprint -rmg adg vol > /tmp/vol
# vxedit -g adg -rf rm vol
# vi /tmp/vol
[…]
        cdsrecover_seqno=0
[…]
# vxmake -g adg -d /tmp/vol
# vxvol -g adg init active vol
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c1t1d0   ENA
pl vol-02       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c1t1d1   ENA
```

11.5 Restore of Lost VxVM Objects

11.5.1 vxprint and vxmake Capabilities

Often, the cause of trouble does not stem from the machines, but sits or stands before them. So we need to consider what to do, if we inadvertently removed objects from the VxVM configuration.

Fortunately, we can use a special output format of **vxprint** to store the disk group configuration in a manner to be read by **vxmake** in order to rebuild the objects. The following table shows in a comparison of the syntax, how both commands specify the appropriate VxVM objects:

VxVM Object Type	**vxprint** options	**vxprint** type abbreviations	**vxmake** type abbreviations
Volume	**-v**	**v**	**vol**
Plex	**-p**	**pl**	**plex**
Subdisk	**-s**	**sd**	**sd**
Subvolume	**-s**	**sv**	**sd**
Data Change Object	**-c**	**dc**	**dco**
Snap Object	**-c**	**sp**	**-**
Cache Object	**-C**	**co**	**cache**
Subcache	**-s**	**sc**	**sd**
Replicated Volume Group	**-V**	**rv**	**rvg**
Remote Link	**-P**	**rl**	**rlink**
Volume Set	**-x**	**vt**	**vset**

The **vxprint** command is able to store all disk group configuration data together with hardware related information, such as the current disk device access path. Unfortunately, the corresponding **vxmake** command cannot define disk and disk group attribute values (therefore not shown in the table above). So we take in mind two restrictions: First, we must initialize the required disks and create the disk group out of them with the well-known commands, before we can read the still missing disk group configuration into **vxmake**. Furthermore, we need to eliminate any disk and disk group information before sending the stored configuration to **vxmake**.

To begin with the latter task: To print all disk group configuration data, including the disk and disk group objects, **vxprint** provides two options: **-a** to print attribute-value pairs word by word, one terribly long line for each object. This output is useful for scripting purposes, because Unix provides many tools to split into lines and words (e.g. **awk/nawk/gawk**, the Shell itself), but rarely into separate paragraphs. The other option, **-m**, creates an output containing the same information, but differently formatted: attribute-value pairs line by line, grouped together to sometimes long paragraphs (only the first line to an object is not indented).

The output of **vxprint -m** is understood by **vxmake**. We still need to skip disk and disk group data by choosing appropriate options. This is much more comfortable than generating the whole output and then erasing the wrong lines by using an editor. Recalling the options table above, we enter the following command:

```
# vxprint -g <diskgroup> -m -vpscCVPx > <configfile>
```

11.5.2 Restore of All Volumes in a Disk Group

If necessary, we initialize disks and recreate the empty disk group manually. We must take care, that the position and length of the private region corresponds to the former settings. Otherwise, the subdisks would be recreated at a different position on the disks, thus providing unusable volume data and possibly failing in creating subdisks at a position outside of the physical disk, if the private region becomes larger than the former one. Then, we read the stored configuration into **vxmake**:

```
# vxmake -g <diskgroup> -d <configfile>
```

That is already nearly all! All disk group objects are recreated at exactly the same physical and virtual location and with the same attribute values. The single exception (we might expect this from our knowledge to the **vxmake** command): Volumes and plexes are still stopped in the state pair DISABLED/EMPTY. To start the volumes together with read-writeback mirror or RAID5 plex synchronization, simply issue:

```
# vxvol -g <diskgroup> startall
```

To skip synchronization (the disk group was cleanly deported, the volumes were stopped, or we do not care for any previous data):

```
# vxvol -g <diskgroup> init active <volume-list>
```

Some pages before, we demonstrated other ways to specify the synchronization method against the assumptions of VxVM. You may apply them as well.

11.5.3 Restore of Some Volumes in a Disk Group

In case of a lost volume, while the disk group and other objects are still configured and alive, we must filter only those objects and attributes belonging to that volume from the stored configuration file. First the good news: **vxprint** is capable to read the disk group configuration not only from the kernel (the default), not only from the physical private region (**/usr/lib/vxvm/diag.d/vxprivutil**), but also from a configuration file in the **vxprint -m** output format. We only add the option **-D** together with the source. Currently only STDIN, specified with the minus sign, is implemented. Thanks to the Shell, we redirect STDIN to the configuration file (the following two commands are identical in result):

```
# vxprint -rLtg <diskgroup> -D - < <configfile>
# cat <configfile> | vxprint -rLtg <diskgroup> -D -
```

Of course, any other output formatting options besides **-rLt** used above are allowed, even **-m**. So, by specifying the volume name together with the "related" option **-r**, we finally get the desired filtered volume output to be piped into **vxmake** (instead of a configuration file, **vxmake -d** understands the minus sign as its argument, now reading from STDIN):

```
# vxprint -rmg <diskgroup> -D - <volume> < <configfile> |
  vxmake -g <diskgroup> -d -
```

We mentioned "good news", because there are, unfortunately, also bad news: This powerful filtering procedure currently does not work in VxVM 5.0 (VxVM 4.x is fine), it produces a core dump due to segmentation fault (jump into not included memory regions). As an acceptable work-around for VxVM 5.0, we propose to store the disk group configuration twice. First, the whole disk group configuration data (except for disks and disk group, of course):

```
# vxprint -g <diskgroup> -m -vpscCPVx > <configfile>.<diskgroup>
```

And, additionally, all volume configuration data, volume by volume, into different configuration files:

```
# for Volume in $(vxprint -g <diskgroup> -vne 'v_layered=off && v_islog=no'); do
  vxprint -rmg <diskgroup> $Volume > <configfile>.<diskgroup>.$Volume; done
```

To restore an inadvertently removed volume, use the volume specific configuration file without further filtering by the **vxprint** command:

```
# vxmake -g <diskgroup> -d <configfile>.<diskgroup>.<volume>
```

The manual page of **vxmake** also mentions another configuration format understood by **vxmake**. This format exactly corresponds to the command line options of **vxmake**, excluding the **vxmake** command itself and the disk group option **-g** with its argument. Since there is no easy way to store the current disk group configuration using this layout, we skip further details.

11.5.4 Restore of the Entire Disk Group Configuration

The boot process (**/etc/rc2.d/S95vxvm-recover** or **/lib/svc/method/vxvm-recover**) starts another VxVM daemon of some interest for disk group disaster scenarios:

```
# ptree $(pgrep -xu0 vxconfigbackupd)
501   /sbin/sh - /usr/lib/vxvm/bin/vxconfigbackupd
  1323  /sbin/sh - /usr/lib/vxvm/bin/vxconfigbackupd
    1324  vxnotify
```

Comparable to the **vxrelocd/vxnotify** daemon pair, **vxconfigbackupd** is informed by **vxnotify** about every VxVM event, even about small configuration changes. See the following example: **vxnotify** is started on one terminal, while we execute a simple plex attach to a volume on another terminal.

```
# vxnotify
connected
[…]
# vxassist -g adg mirror vol
# vxnotify        (continued)
change dg adg dgid 1223834080.19.sols
change dg adg dgid 1223834080.19.sols
change dg adg dgid 1223834080.19.sols
^C
```

The first message belongs to the creation of a new plex together with its subdisk(s), the second indicates the begin, the last the end of plex synchronization.

Now, **vxconfigbackupd** captures the output of **vxnotify** and triggers in case of configuration change events a new backup of the new (!) configuration. You will find the backups under **/etc/vx/cbr/bk**:

```
# cd /etc/vx/cbr/bk
# ls -l
drwxr-xr-x  2 root    root       1536 Jun 12 12:25 adg.1213178377.21.sols
drwxr-xr-x  2 root    root       1536 Oct 12 21:14 adg.1223834080.19.sols
drwxr-xr-x  2 root    root       1536 Oct 12 10:08 osdg.1198162173.6.sols
```

Each subfolder is named after the name and the ID of the related disk group. Concerning the disk group **adg**, we recognize two versions. Remember: Only the disk group ID is unique. Obviously, the disk group **adg** was once unavailable and freshly created under the same name, but naturally not with the same disk group ID. The content of the latter **adg** folder is (given VxVM 5.0):

```
# cd adg.1223834080.19.sols
# ls -l
-rw-r--r--  1 root    root     655360 Oct 12 21:14 1223834080.19.sols.binconfig
-rw-r--r--  1 root    root       8035 Oct 12 21:14 1223834080.19.sols.cfgrec
-rw-r--r--  1 root    root       1695 Oct 12 21:14 1223834080.19.sols.dginfo
-rw-r--r--  1 root    root       3190 Oct 12 21:14 1223834080.19.sols.diskinfo
```

The first file with extension **binconfig** is nearly a physical copy of the disk and disk group configuration part of the private region. The next file (**cfgrec**) contains the output of **vxprint -mg adg** (including disk and disk group objects and attributes), the third (**dginfo**) some VxVM version and particularly disk group information, the last file (**diskinfo**) details to the disk group members.

Note the large size of the private region copy file. Our disk group example deals with private regions only 1 MB in size. In VxVM 5.0, the default size of the private region is 32 MB, so in enterprise environments providing many disk groups the **binconfig** files occupy large amounts of storage of the root file system. That's why only the last known configuration is stored by VxVM 5.0, while VxVM 4.x holds the last six versions as a source to recover former disk group configurations.

Well, that is a nice thing. We just prepare the disk group in order to get some experience on the procedure to recover a disk group. Unfortunately, a simple **vxdg destroy** command will also remove the backup of the disk group configuration (there are other ways to recreate a destroyed disk group, see below). So we temporarily rename the backup folder and make sure that no private region data exist anymore on the disks.

```
# cd /etc/vx/cbr/bk
# ls -d adg.*
```

```
adg.1213178377.21.sols  adg.1223834080.19.sols
# mv adg.1223834080.19.sols adg.1223834080.19.sols.renamed
# vxdg destroy adg
# mv adg.1223834080.19.sols.renamed adg.1223834080.19.sols
# vxdiskunsetup c4t1d0
# vxdiskunsetup c4t2d0
# dd if=/dev/zero of=/dev/rdsk/c4t1d0s2 bs=128k oseek=1 count=8
# dd if=/dev/zero of=/dev/rdsk/c4t2d0s2 bs=128k oseek=1 count=8
# /usr/lib/vxvm/diag.d/vxprivutil dumpconfig /dev/rdsk/c4t1d0s2
VxVM vxprivutil ERROR V-5-1-1735 scan operation failed:
        Format error in disk private region
# /usr/lib/vxvm/diag.d/vxprivutil dumpconfig /dev/rdsk/c4t2d0s2
VxVM vxprivutil ERROR V-5-1-1735 scan operation failed:
        Format error in disk private region
```

Are you convinced that neither in the kernel memory nor on the disks any usable disk group data are available? Our single hope lies in the configuration backup. Look at the following commands, how to (and how not to) reactivate the disk group (the option **-p** signifies "precommit").

```
# vxconfigrestore -p adg
VxVM vxconfigrestore ERROR V-5-2-3717 There are two backups that have the same
diskgroup name adg with different diskgroup id :
  1213178377.21.sols -- backup at Thu Jun 12 12:25:42 MEST 2008
  1223834080.19.sols -- backup at Sun Oct 12 21:14:26 MEST 2008

VxVM vxconfigrestore INFO V-5-2-3721 Use diskgroup_id to do the restoration.
# vxconfigrestore -p 1223834080.19.sols
Diskgroup 1223834080.19.sols configuration restoration started ......

Installing volume manager disk header for c4t1d0s2 ...
c4t1d0s2 disk format has been changed from cdsdisk to none.
Installing volume manager disk header for c4t2d0s2 ...
c4t2d0s2 disk format has been changed from cdsdisk to none.

1223834080.19.sols's diskgroup configuration is restored (in precommit state).
Diskgroup can be accessed in read only and can be examined using
vxprint in this state.

Run:
  vxconfigrestore -c 1223834080.19.sols ==> to commit the restoration.
  vxconfigrestore -d 1223834080.19.sols ==> to abort the restoration.
```

We notice that the disks of the disk group were initialized to some extent (whatever **volume manager disk header** means, see below; note also the erroneously inverted direction in the expression **from cdsdisk to none**). Commands exploring kernel information on the disk group work as usual, even modification of the configuration is possible, obviously unrestricted (including mirror synchronization):

```
# vxdisk -g adg list
DEVICE       TYPE            DISK         GROUP        STATUS
c4t1d0s2     auto:cdsdisk    adg01        adg          online
c4t2d0s2     auto:cdsdisk    adg02        adg          online
# vxprint -rtg adg
[…]
v  vol          -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c4t1d0   ENA
pl vol-02       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c4t2d0   ENA
# vxassist -g adg -b make newvol 1g layout=mirror
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   177        RDWRBACK/R 44.43% 0/2097152/931840 VOLSTART newvol adg
```

So what do they mean with the expression **Diskgroup can be accessed in read only** in the output of **vxconfigrestore**? Well, read access to the volumes is possible, but write access is blocked. Therefore, you may analyze the content of the volumes in order to check the proper position of the subdisks.

```
# dd if=/dev/vx/rdsk/adg/vol of=/dev/null count=1
1+0 records in
1+0 records out
# dd if=/dev/zero of=/dev/vx/rdsk/adg/vol count=1
write: Permission denied
1+0 records in
1+0 records out
```

What became of the physical state of the disks? Well, the partitions for VxVM were recreated. In VxVM 4.x (our example), the private region got initialized, but just with the disk group and the disk media object on disks formerly being inactive configuration disks (**c4t1d0s2**). In addition, volume related objects are written to the private region of previously active config disks (**c4t2d0s2**). VxVM 5.0 does not write any disk group configuration on the disks at this stage, working just with the kernel configuration.

```
# prtvtoc -h /dev/rdsk/c4t1d0s2
       2      5    01          0  35368272  35368271
       7     15    01          0  35368272  35368271
# prtvtoc -h /dev/rdsk/c4t2d0s2
       2      5    01          0  35368272  35368271
       7     15    01          0  35368272  35368271
# /usr/lib/vxvm/diag.d/vxprivutil dumpconfig /dev/rdsk/c4t1d0s2 |
  awk '!/^[ #]/ && NF>0'
VxVM vxconfigdump ERROR V-5-1-624 Error (File block 16): Invalid magic number
```

```
dg   adg
dm   adg01
# /usr/lib/vxvm/diag.d/vxprivutil dumpconfig /dev/rdsk/c4t2d0s2 |
  awk '!/^[ #]/ && NF>0'
dg   adg
dm   adg01
dm   adg02
plex vol1-01
dm   adg03
sd   adg02-01
sd   adg03-01
vol  vol1
plex vol1-02
sd   adg01-02
```

As indicated by the output of the command **vxconfigrestore -p**, the disk group configuration must be committed (**-c**) or discarded (**-d**). Assuming a proper disk group configuration, we commit it. Redundant volumes are synchronized (even **newvol** once again) in order to ensure data integrity. Any requests to modify synchronization behavior may be achieved by modification of the plex states during the precommit stage (see the examples beginning on page 383).

```
# vxconfigrestore -c 1223834080.19.sols
Committing configuration restoration for diskgroup 1223834080.19.sols ....

1223834080.19.sols's diskgroup configuration restoration is committed.
# vxprint -rtg adg newvol
[…]
v  newvol      -           ENABLED  SYNC     2097152  SELECT    -       fsgen
pl newvol-01   newvol      ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg01-01    newvol-01   adg01    0        2097152  0         c4t1d0  ENA
pl newvol-02   newvol      ENABLED  ACTIVE   2097152  CONCAT    -       RW
sd adg02-01    newvol-02   adg02    0        2097152  0         c4t2d0  ENA
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   179         RDWRBACK/R 89.06% 0/2097152/1867776 VOLSTART newvol adg
```

11.5.5 Restore of a Destroyed Disk Group

VxVM protects against most erroneous actions destroying volume data. Dissociating a plex (that is, an instance of the volume address space) from its volume will fail, if this plex was the last healthy plex in the volume (unless you specified **vxplex -o force dis**). All the same, you cannot dissociate a subdisk from its plex except by **vxsd -o force dis**. Removing a volume by VxVM interfaces is absolutely impossible as long as the volume attribute **devopen** is set to **on** (current I/O on raw device or file system mounted).

Nevertheless, it happens from time to time (hopefully not too often) that you destroyed

VxVM objects you wanted or should have wanted to keep. The former sections explained and demonstrated how to recreate them out of configuration files and how to restore a disk group, if their (active configuration) disks were destroyed. Our task now is to provide a way to recover from an inadvertently destroyed disk group configuration.

```
# vxdisk -g adg list
DEVICE       TYPE            DISK         GROUP        STATUS
c1t1d0s2     auto:cdsdisk    adg01        adg          online
c1t1d1s2     auto:cdsdisk    adg02        adg          online
# vxdg destroy adg
# vxdisk -o alldgs list
DEVICE       TYPE            DISK         GROUP        STATUS
[…]
c1t1d0s2     auto:cdsdisk    -            -            online
c1t1d1s2     auto:cdsdisk    -            -            online
[…]
```

Obviously, the disk group has gone, at least for the **vxdisk list** tool. Let's have a closer look to one of the disks formerly belonging to the destroyed disk group.

```
# vxdisk -s list c1t1d0
[…]
diskid: 1223448582.74.haensel
dgname:
dgid:   1223448150.61.haensel
hostid:
info:   format=cdsdisk,privoffset=256,pubslice=2,privslice=2
```

That looks more encouraging than we expected. The disk group name indeed was erased from the disk configuration stored in the private region together with the disk group ownership (field **hostid**), but the disk group ID providing a unique identification in case of disk group name collisions is still available. In order to determine the amount of disk group configuration data still present, we analyze the content of the private region (converted into a well arranged output format).

```
# /usr/lib/vxvm/diag.d/vxprivutil dumpconfig /dev/rdsk/c1t1d0s2 | vxprint -rtD -
v  vol          -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol-01       vol          DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         -        DIS
pl vol-02       vol          DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         -        DIS
```

A full comparison of the unconverted output with the last disk group configuration stored by **vxprint -m** would reveal that none of these attributes has gone. Only those two attributes of the **disk configuration** mentioned above are lost: the disk group name and the disk group ownership. Well, that can easily be recreated by importing the disk group using the disk group ID. You do not need to specify the disk group name, for it is automatically taken from the **disk group configuration**.

```
# vxdg import 1223448150.61.haensel
# vxdisk -g adg list
DEVICE       TYPE            DISK         GROUP        STATUS
c1t1d0s2     auto:cdsdisk    adg01        adg          online
c1t1d1s2     auto:cdsdisk    adg02        adg          online
```

To sum up: Identify at least one disk formerly belonging to the destroyed disk group (the disk may not provide an active configuration copy of the disk group). Get the disk group ID from the disk configuration and import the disk group by its ID. That's all!

Did you shiver when reading about the task of disk identification in the previous paragraph? Indeed, this could be extremely complicated if you administer a large environment with numerous disks and if you do not remember the device access of at least one disk. Searching for other locations of stored disk group configuration files, we draw a blank. The disk group configuration copy created by **vxconfigbackupd** under **/etc/vx/cbr/bk** was removed as a result of the disk group destruction. Only **vxrelocd** and **vxsparecheck** provide an automated configuration backup under **/etc/vx/saveconfig.d**, which survives even disk group destruction. But very probably, you will not get a disk outage ready at hand to get the necessary backup file created in order to manage an inadvertent disk group destruction.

We provide a dirty procedure to map the disk group name to its disk group ID and to identify a disk formerly belonging to the destroyed disk group — dirty, for it is undocumented. A file named after the disk group in a subdirectory of **/var**, designed to store the temporary utility attribute values to **tutil0**, **tutil1**, and **tutil2**, has kept information on object names even of the destroyed disk group. The content is, however, somewhat coded, but fortunately, Unix provides valuable tools to filter printable characters:

```
# file /var/vxvm/tempdb/adg
/var/vxvm/tempdb/adg:    data
# strings -n 3 /var/vxvm/tempdb/adg | grep -v VBLK
VMDB
adg
1223448150.61.haensel
adg
adg01
adg02
adg01-01
adg02-01
vol
vol-01
vol-02
```

The line containing the disk group name is immediately followed by the specification of the disk group ID. Our task, the mapping of the disk group name to its ID, was successful.

As a procedure of last resort, you must analyze all available disks obviously not belonging to an active disk group. Check for the existence and value of the disk group ID stored in the disk configuration part of the private region and filter from the disk group configuration part the name of the disk group. We provide a short **ksh** script displaying device access name, disk group name, and disk group ID for all appropriate disks. The device access name may be helpful in case of multiple instances of a disk group name (same name, but different disk group IDs).

```
# List all disks being no disk group member and print device access
vxdisk -qo alldgs list | awk '$4=="-" {print $1}' |
while read daname; do
    # Get disk group ID, if any
    DGID=$(vxdisk -s list $daname | awk '$1=="dgid:" && NF==2 {print $2}')
    # Next disk, if no disk group ID
    [[ -z $DGID ]] && continue
    # Print device access
    printf '%-12s ' $daname
    # Dump disk group configuration of disk's private region
    /usr/lib/vxvm/diag.d/vxprivutil dumpconfig /dev/vx/rdmp/$daname |
    # Filter name and disk group ID from disk group object
    vxprint -D - -GF '%{name:-12} %dgid'
done
```

An example output of two destroyed disk groups:

```
c1t1d0s2     adg          1223278682.30.haensel
c1t1d1s2     adg          1223278682.30.haensel
c1t1d2s2     adg          1223278682.30.haensel
c1t1d3s2     adg          1223278682.30.haensel
c1t1d4s2     bdg          1223278792.32.haensel
c1t1d5s2     bdg          1223278792.32.haensel
```

11.5.6 Serial Split Brain of a Disk Group

In general, it is a good idea to mirror your volumes across two (or more) separated locations, for all volumes go on to provide data access in case of a data center outage. It is also a good idea to improve the availability of your applications by putting them under control of a clustering software (a "campus cluster").

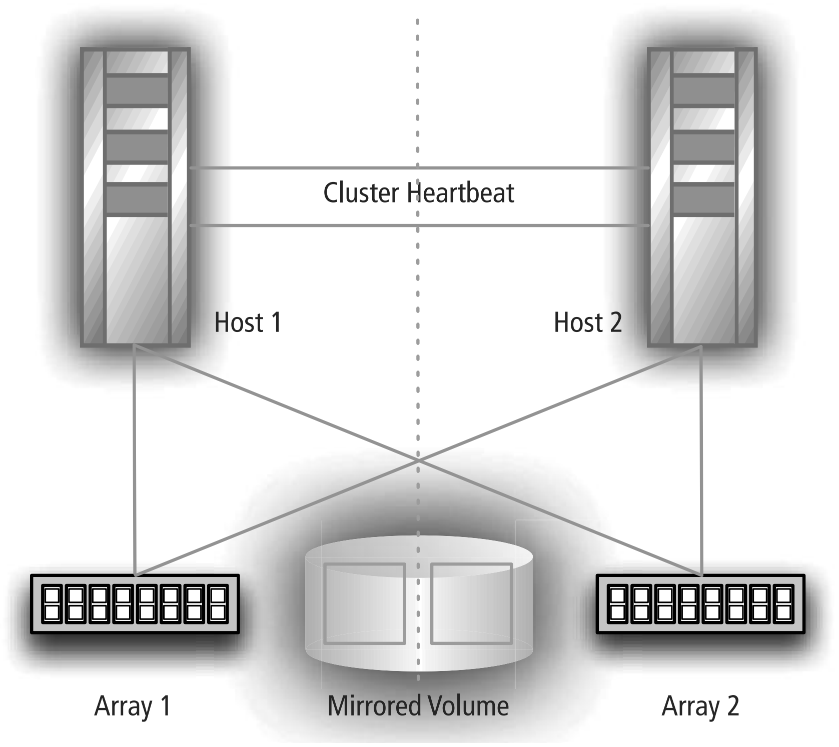


Figure 11-3: A Mirrored Volume in a Campus Cluster

Nevertheless, this architecture includes a new problem in case of a network partition: Each cluster host believes being the only surviving cluster node, thus starting all those applications that were (and still are) running on the other node. We enter the world of the dreaded cluster split brain. Loss of application data consistency is now a matter of seconds.

However, our current topic is not application data consistency. As long as no disk group configuration changes are performed, the disk group remains consistent across all nodes. But what happens if both nodes modify the disk group? Then, no modified version can be considered the current one by VxVM, administrator intervention is required. The VxVM documentation calls this scenario a "Serial Split Brain" (SSB). Our example below will simulate it by a sequential removal of the private region content from both disks of a disk group.

1. The disk group configuration contains a simple volume mirrored across both disks of the disk group.

```
# vxprint -rtg adg
[…]
```

```
dm adg01        c4t1d0s2     auto     2048     35365968 -
dm adg02        c4t2d0s2     auto     2048     35365968 -

v  vol          -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol-01       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c4t1d0   ENA
pl vol-02       vol          ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         c4t2d0   ENA
```

2. We store the private region content of the first disk and clear it afterwards. In order to inform the kernel on the damaged disk state, we deport and forcibly re-import the disk group. Finally, we modify the disk group configuration by adding a new volume.

```
# dd if=/dev/rdsk/c4t1d0s2 of=/var/tmp/c4t1d0s2 bs=128k iseek=1 count=8
# dd if=/dev/zero of=/dev/rdsk/c4t1d0s2 bs=128k oseek=1 count=8
# vxdg deport adg
# vxdg -f import adg
VxVM vxdg WARNING V-5-1-560 Disk adg01: Not found, last known location: c4t1d0s2
# vxassist -g adg make newvol 1g
```

3. The following commands simulate the view on the disk group from another cluster node: The disk group is deported, its previous configuration restored on the first disk, while the second disk's private region is temporarily overwritten. A forced re-import indeed shows the old configuration (unlike a real cluster scenario, the volume is DISABLED due to the re-import). Once again, we modify the disk group by adding a volume.

```
# vxdg deport adg
# dd if=/var/tmp/c4t1d0s2 of=/dev/rdsk/c4t1d0s2 bs=128k oseek=1
# dd if=/dev/rdsk/c4t2d0s2 of=/var/tmp/c4t2d0s2 bs=128k iseek=1 count=8
# dd if=/dev/zero of=/dev/rdsk/c4t2d0s2 bs=128k oseek=1 count=8
# vxdg -f import adg
VxVM vxdg WARNING V-5-1-560 Disk adg02: Not found, last known location: c4t2d0s2
# vxprint -rtg adg
[…]
dm adg01        c4t1d0s2     auto     2048     35365968 -
dm adg02        -            -        -        -        NODEVICE

v  vol          -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol-01       vol          DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         c4t1d0   ENA
pl vol-02       vol          DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         -        NDEV
# vxassist -g adg make newvol 1g
```

4. Any clean-up procedure should start by stopping the application and deporting the disk group in order to discard any locally stored data and configurations. Regarding the disk group configuration, our simulation needs to recreate the private region content

on the second disk. Currently, no configuration copy may be interpreted as a newer version compared to the other one. VxVM should not and indeed will not decide in favor of one copy, when detecting the "Serial Split Brain" by trying to import the disk group.

```
# vxdg deport adg
# dd if=/var/tmp/c4t2d0s2 of=/dev/rdsk/c4t2d0s2 bs=128k oseek=1
# vxdg import adg
VxVM vxdg ERROR V-5-1-587 Disk group adg: import failed: Serial Split Brain
detected. Run vxsplitlines
```

5. We observe the recommendation of the last command output. **vxsplitlines** informs us about different configuration copies, where they are stored, how to access them in order to decide for one copy (**vxprivutil dumpconfig**), and how to restore the disk group based on our copy selection (**vxdg -o selectcp=1223115776.23.sols import**). The copy selection is based on the disk ID assigned by VxVM during disk initialization.

```
# vxsplitlines -g adg
  VxVM vxsplitlines NOTICE V-5-2-2708 There are 2 pools.
The Following are the disks in each pool. Each disk in the same pool
has config copies that are similar.
  VxVM vxsplitlines INFO V-5-2-2707 Pool 0.
c4t2d0s2 adg02
To see the configuration copy from this disk issue
/etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c4t2d0s2
To import the diskgroup with config copy from this
disk use the following command
/usr/sbin/vxdg -o selectcp=1223115776.23.sols import adg
The following are the disks whose ssb ids don't match in this config
copy
adg01

  VxVM vxsplitlines INFO V-5-2-2707 Pool 1.
c4t1d0s2 adg01
To see the configuration copy from this disk issue
/etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c4t1d0s2
To import the diskgroup with config copy from this
disk use the following command
/usr/sbin/vxdg -o selectcp=1223014683.15.sols import adg
The following are the disks whose ssb ids don't match in this config
copy
adg02
```

6. We analyze the content of the private regions on the disks (using the DMP drivers). The default output format of **vxprivutil dumpconfig** is similar to the output generated by **vxprint -m** (except for comments on device blocks). Mostly, we do not need to check so many details of the disk group. Therefore, at least for demonstration purposes on commands working neatly together, we pipe the output into a reformatting **vxprint** com-

mand. Indeed, the subdisk location of **newvol** differs, as emphasized in the output.

```
# /etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c4t1d0s2 | vxprint -rtD -
Disk group: adg
[…]
v  vol          -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol-01       vol          DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         -        DIS
pl vol-02       vol          DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         -        DIS

v  newvol       -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl newvol-01    newvol       DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg01-02     newvol-01    adg01    2097152  2097152  0         -        DIS
# /etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c4t2d0s2 | vxprint -rtD -
Disk group: adg
[…]
v  vol          -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol-01       vol          DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        2097152  0         -        DIS
pl vol-02       vol          DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        2097152  0         -        DIS

v  newvol       -            DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl newvol-01    newvol       DISABLED ACTIVE   2097152  CONCAT    -        RW
sd adg02-02     newvol-01    adg02    2097152  2097152  0         -        DIS
```

7. Just to make sure, that the command output of **vxsplitlines** identified the disks properly, we compare it to the disk IDs. Then, we import the disk group based on our selected disk group configuration copy (**newvol**'s subdisk on disk **adg01**).

```
# vxdisk -s list c4t1d0
[…]
diskid: 1223014683.15.sols
[…]
# vxdisk -s list c4t2d0
[…]
diskid: 1223115776.23.sols
[…]
# vxdg -o selectcp=1223014683.15.sols import adg
```

8. Being somewhat distrustful given our experience with software products for many years, we check for the result and perform a detailed comparison of the physical contents of the private region copies. Hooray, they are indeed identical regarding the disk group configuration!

```
# vxprint -rtg adg
```

```
[…]
dm adg01       c4t1d0s2    auto     2048     35365968 -
dm adg02       c4t2d0s2    auto     2048     35365968 -

v  vol         -           DISABLED ACTIVE   2097152  SELECT   -       fsgen
pl vol-01      vol         DISABLED ACTIVE   2097152  CONCAT   -       RW
sd adg01-01    vol-01      adg01    0        2097152  0        c4t1d0  ENA
pl vol-02      vol         DISABLED ACTIVE   2097152  CONCAT   -       RW
sd adg02-01    vol-02      adg02    0        2097152  0        c4t2d0  ENA

v  vola        -           DISABLED ACTIVE   2097152  SELECT   -       fsgen
pl vola-01     vola        DISABLED ACTIVE   2097152  CONCAT   -       RW
sd adg01-02    vola-01     adg01    2097152  2097152  0        c4t1d0  ENA
# /etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c4t1d0s2 > /tmp/c4t1
# /etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c4t2d0s2 > /tmp/c4t2
# diff /tmp/c4t[12]        (no output, i.e. no difference)
```

11.6 Booting without VxVM

A heavily corrupted VxVM installation could lead to an unbootable system. At least two steps within the boot process could fail.

1. Kernel initialization and configuration are based on the root device. The root device is the Open Boot Prom device providing access to the physical partition that holds the root file system. The root file system may not be a unique device, your system naturally provides in most cases two or more, because mostly you have mirrored your OS devices. But even one instance of the root file system may be accessed by different drivers.

By the Open Boot Prom disk driver:

```
/ssm@0,0/pci@18,700000/pci@1/scsi@2/disk@2,0:a
```

As a pseudo device, by the VxVM kernel module responsible for the conversion of application I/O requests into disk I/Os:

```
/pseudo/vxio@0:0
```

Without a **rootdev** entry in the **/etc/system** file, the boot device (such as **/ssm@0,0/pci@18,700000/pci@1/scsi@2/disk@2,0:a**) automatically serves as root device (by default, **/etc/system** under Solaris only contains comments). For the need to start VxVM step by step, already the root device must be redirected to the VxVM kernel module **vxio**. Therefore, a working boot process based on VxVM needs two active lines in the file **/etc/system**:

```
rootdev:/pseudo/vxio@0:0
set vxio:vol_rootdev_is_volume=1
```

The latter line sets a memory flag indicating to create a virtual volume on the root device (**vol_rootdev_is_volume** is a symbolic memory address name). If you like kernel panics in an early stage of the boot process, just feel free to corrupt any of these two lines. But if both lines are missing, the boot process chooses the boot device as the root device, thus building and configuring the kernel based on the standard disk driver.

2. Even if the file **/etc/system** does not contain the required entries for a VxVM encapsulated OS disk, so that a partition based boot process is initiated, we must deal with another barrier: The file **/etc/vfstab** still contains volume drivers. Quite early in the single user mode/milestone, the required OS file systems /, **/usr**, and **/var** are mounted or remounted in read-write mode based on the entries in **/etc/vfstab**. But the volume drivers specified in this file will fail in case of a corrupted VxVM installation. Therefore, we need a file version containing partition drivers.

In order to be well prepared for the recovery of a completely corrupted VxVM installation, any bootable disk should contain a partition bootable and a volume bootable version of **/etc/system** and **/etc/vfstab** in addition to the active files. Quite simple for **/etc/system**:

```
# cp /etc/system /etc/system.vol
# cp /etc/system /etc/system.part
# vi /etc/system.part     (commented out by asterisk followed by space or completely erased)
[…]
* rootdev:/pseudo/vxio@0:0
* set vxio:vol_rootdev_is_volume=1
[…]
```

Creating a partition based **/etc/vfstab** for each bootable disk is a lot more complicated. We need a mapping of the mount points to the numbers of the partitions covering the subdisks of our OS volumes on our disks. Actually, we will settle for the volume names, assuming that they correspond to the default names (**rootvol** mounted on /, **swapvol** for the swap device, **<volume>** mounted on ***/<volume>**). An approach step by step! The subdisk offsets are calculated against the beginning of the public region (do not be alarmed by the ghost subdisk **osdg01-B0**, we will take it in mind):

```
# vxprint -g bootdg -sF '%name %dm_offset'
osdg01-B0 78083039
osdg01-01 0
osdg01-02 4198319
[…]
```

On the other side, the partition offsets are calculated against the beginning of the disk.

```
# prtvtoc -h /dev/rdsk/c0t0d0s2
       0      2    00    4198320  12586800  16785119
       1      3    01          0   4198320   4198319
       2      5    00          0  78156480  78156479
```

[…]

Therefore, the subdisk offsets plus the offset of the public region on the disk will result in the partition offsets. Where do we get the public region offset from?

```
# vxdisk list c0t0d0s2 | nawk '$1=="public:"'
public:    slice=3 offset=1 len=78083040 disk_offset=0
```

In the output above, **disk_offset** specifies the offset of the public region **as partition** relative to the disk beginning, while **offset** denotes the offset of the public region **as subdisk container** relative to the beginning of the public region **as partition**. So, we must add **offset** and **disk_offset** to the subdisk offsets in order to get the subdisk offsets relative to the disk beginning which are, of course, identical to the partition offsets. We erase the key words together with their equal sign from the output above and add the resulting third to the fifth word to get the public region offset **as subdisk container** to the beginning of the disk:

```
# vxdisk list c0t0d0s2 |
  nawk '$1=="public:" {gsub(/[^ =]+=/,""); print $3+$5}'
1
```

Just one exception: If a partition starts at sector 0, it contains the VTOC at the beginning. To protect the VTOC, the public region starting at sector 0 **as partition** gets an **offset** of one sector **as subdisk container** (see the Encapsulation chapter, page 330). Our calculation above would produce a negative value (-1) for the corresponding subdisk. In that case, we modify the value to 0 and fix the miscalculation due to the ghost subdisk. Here is our script with some comments, but without error checking:

```
# vi map_vol_part
#!/bin/ksh
# c#t#d#s2 syntax is required
Disk=$1

# Get disk offset of public region as subdisk container
PubRegOffset=$(vxdisk list $Disk |
nawk '$1=="public:"{gsub(/[^ =]+=/,""); print $3+$5}')

prtvtoc -h /dev/rdsk/$Disk |
# Skip backup, private and public region partition
nawk '$2!=5 && $2!=14 && $2!=15 {print $1,$4}' |
while read Slice DiskOffset; do
    ((SubdiskOffset=DiskOffset-PubRegOffset))
    # Handle VTOC protection by ghost subdisk
    ((SubdiskOffset==-1)) && ((SubdiskOffset=0))
    # Get plexes to the corresponding subdisks on the disk
    # and print associated volume name
    Volume=$(vxprint -g bootdg -pF %assoc -e \
    "pl_sd.sd_dm_offset=$SubdiskOffset && pl_sd.sd_da_name=\"$Disk\"")
    printf '%-12s %s\n' $Volume $Slice
done
# chmod 744 map_vol_part
# ./map_vol_part c0t0d0s2
rootvol      0
swapvol      1
var          5
opt          6
# ./map_vol_part c0t2d0s2
rootvol      0
swapvol      1
opt          5
var          6
```

Finally, we got the data required to create vfstabs for each bootable disk. Look at the example based on Solaris 9 (Solaris 10 adds three more lines for **/devices**, **ctfs**, and **objfs**):

```
# vi /etc/vfstab.c0t0d0s2
#device             device               mount     FS     fsck  mount    mount
#to mount           to fsck              point     type   pass  at boot  options
fd                  -                    /dev/fd   fd       -   no       -
/proc               -                    /proc     proc     -   no       -
swap                -                    /tmp      tmpfs    -   yes      -
/dev/dsk/c0t0d0s0   /dev/rdsk/c0t0d0s0   /         ufs      1   no       logging
/dev/dsk/c0t0d0s1   -                    -         swap     -   no       -
/dev/dsk/c0t0d0s5   /dev/rdsk/c0t0d0s5   /var      ufs      1   no       logging
/dev/dsk/c0t0d0s6   /dev/rdsk/c0t0d0s6   /opt      ufs      1   yes      logging
```

Be sure to create vfstabs for ALL your bootable disks in order to be prepared for an unexpected system-wide VxVM failure. Did you ever use the **vi** editor at an early stage of the single user mode/milestone without a **/tmp** device, thus only with single line editing capabilities? Well, then you will appreciate our hints:

```
# cp /etc/vfstab /etc/vfstab.vol
# vi /etc/vfstab.c0t0d0s2
[…]
# vi /etc/vfstab.c0t2d0s2
[…]
```

Since we are now prepared for the worst (a Solaris system without working VxVM), let's test the procedure to boot for once based on partitions. Our VxVM disaster simulation is created by renaming the basic kernel module **vxio**.

```
# mv /kernel/drv/sparcv9/vxio /kernel/drv/sparcv9/vxio.renamed
```

A regular reboot will fail due to the wrong files **/etc/system** and **/etc/vfstab**, as explained above. We could easily copy the proper files before rebooting, but this would not fit to an unexpected VxVM disaster. Therefore, without any system reconfiguration, we shut down our system to the Open Boot Prom.

```
# init 0          (or halt)
```

The Open Boot Prom provides a mechanism to boot from alternate devices and kernel configurations. That's exactly what we need: a kernel configuration based on a different **/etc/system** (still not part of the OBP device tree, therefore without leading slash below). Other requested parameter defaults simply require confirmation. Further on, we just enter the single user mode/milestone, skipping even the execution of its run-level scripts, for our basic VxVM kernel module is corrupted.

```
ok boot -ab
Boot device: /pci@1f,0/ide@d/disk@0,0:a  File and args: -ab
Enter filename [kernel/sparcv9/unix]: enter
Enter default directory for modules [/platform/SUNW,Sun-Blade-100/kernel /plat-
form/sun4u/kernel /kernel /usr/kernel]: enter
Name of system file [etc/system]: etc/system.part
SunOS Release 5.10 Version Generic_118833-36 64-bit
Copyright 1983-2006 Sun Microsystems, Inc.  All rights reserved.
Use is subject to license terms.
root file system type [ufs]: enter
Enter physical name of root device
[/pci@1f,0/ide@d/disk@0,0:a]: enter
[…]
Root password for system maintenance (control-d to bypass): password
```

In order to activate the partition based files **/etc/system** and **/etc/vfstab**, our root file system must be remounted to achieve read-write access. A proper key table alleviates this task. Since it is of no use to execute the VxVM run-level scripts during the following reboot, we create a touch file designed to skip them.

```
# loadkeys
# awk '$3=="/"' /etc/vfstab.c0t0d0s2
/dev/dsk/c0t0d0s0   /dev/rdsk/c0t0d0s0   /        ufs     1   no       logging
# mount -F ufs -o remount /dev/dsk/c0t0d0s0 /
# cp /etc/system.part /etc/system
# cp /etc/vfstab.c0t0d0s2 /etc/vfstab
# touch /etc/vx/reconfig.d/state.d/install-db
# reboot
```

Our system now boots without the need for interaction based on partitions of the disk **c0t0d0s2** (our example). It is deadly important to keep in mind that our copy and touch actions, the succeeding boot process, and our VxVM repair modified the OS file systems only on **c0t0d0s2**, without the knowledge of VxVM. Therefore, the OS mirrors are now out of sync. But the last known states of the corresponding plexes stored within the private region are ACTIVE, signifying that the mirrors are coherent. By all means, before performing a volume based boot, we must tell VxVM that the plex mirrors on the other OS disk(s) are stale (see the repairing commands below). Otherwise, the default read policy SELECTED, which means ROUND ROBIN given our simple plex layouts, leads to inconsistent and unpredictable results, to a system unable to work properly.

```
# cp /etc/system.vol /etc/system
# cp /etc/vfstab.vol /etc/vfstab
# rm /etc/vx/reconfig.d/state.d/install-db
# mv /kernel/drv/sparcv9/vxio.renamed /kernel/drv/sparcv9/vxio
# vxconfigd -m boot
# PlexList=$(vxprint -g bootdg -pne 'pl_sd.sd_daname!="c0t0d0s2"')
# vxmend -g bootdg off $PlexList
# vxmend -g bootdg on $PlexList
# vxprint -rtg bootdg
[…]
v  rootvol      -            ENABLED  ACTIVE   12586800 ROUND     -        root
pl rootvol-01   rootvol      ENABLED  ACTIVE   12586800 CONCAT    -        RW
sd osdg01-02    rootvol-01   osdg01   4198319  12586800 0         c0t0d0   ENA
pl rootvol-02   rootvol      DISABLED STALE    12586800 CONCAT    -        RW
sd osdg02-03    rootvol-02   osdg02   29367840 12586800 0         c0t2d0   ENA
[…]
# reboot
```

The reboot process will be based upon volumes and start plex synchronization in the background. Fortunately and finally!

Technical Deep Dive

11.7 More than Two OS Mirrors: Emergency Disk

Sometimes it may be desirable to supply OS or application volumes with a third mirror, e.g. to achieve even higher redundancy or an increased read performance due to the usual round robin read policy. But to mention the most important reason for a third plex (in a special state) within OS or application volumes, mirroring data does not protect against some awful data destroying errors. Some examples: You issue the dreaded **rm -r *** command within the wrong directory, the database did not survive a test run with imported data, an OS "patch" made our system unbootable, and so on.

We focus on the procedure to protect OS volumes by adding a third plex in order to form an emergency boot disk. The emergency disk must hold a bootable OS image which is not updated by every I/O, but only in intervals decided by the administrator (manually or automated). How can we create a third OS mirror? The answer seems quite simple: by the same way we did it for the second OS mirror. Indeed? Let's try it!

1. We initialize a new disk and add it to the boot disk group:

```
# vxdisksetup -i c0t4d0 format=sliced privlen=1m
# vxdg -g osdg adddisk osdg03=c0t4d0
# vxdisk -g osdg list
DEVICE       TYPE            DISK         GROUP        STATUS
c0t2d0s2     auto:sliced     osdg01       osdg         online
c0t3d0s2     auto:sliced     osdg02       osdg         online
c0t4d0s2     auto:sliced     osdg03       osdg         online
```

2. We try to add mirrors by the usual **vxmirror** command:

```
# vxmirror -g osdg osdg01 osdg03
VxVM vxmirror ERROR V-5-2-3604 No non-redundant volumes to mirror
```

Oops! The main purpose of **vxmirror** was (and still is) to mirror non-redundant volumes. Mirroring OS volumes is just a special subtask. But all our volumes are still mirrored. We need another procedure.

3. We create the third mirrors using the **vxassist** command, thus avoiding the mirror check mechanisms of the **vxmirror** script. First, we need a list of OS volumes, but only of top-level volumes, not of subvolumes, DCL volumes, or cache volumes. Unfortunately, the development of attributes to the **-e** option of **vxprint** often lags behind the implementation of new object attributes: The volume attribute **layered**, introduced by VxVM 3.0 long

ago, was not activated for **-e** until VxVM 5.0 in the usual manner (**v_layered**). The attribute **iscachevol**, belonging to space optimized snapshots starting with VxVM 4.0, is still not implemented for **-e**. Therefore, make sure by an advanced volume analysis you only mirror the correct volumes. The first of the following commands, being compatible to VxVM 4.x, just skips DCL volumes (you won't expect subvolumes and snapshot cache volumes on OS disks, will you?), the second one subvolumes as well.

```
# vxprint -g osdg -vne 'v_isdcolog=off'
opt
rootvol
swapvol
var
# vxprint -g osdg -vne 'v_isdcolog=off && v_layered=off'
opt
rootvol
swapvol
var
```

4. Taking this output or a partially reduced list of volumes, we write a simple Shell loop to add another mirror to the volumes:

```
# for vol in $(vxprint -g osdg -vne 'v_isdcolog=off && v_layered=off'); do
  vxassist -g osdg mirror $vol osdg03; done
```

Mirroring **rootvol** carrying usage type **root** automatically creates a partition number 0 over the mirror subdisk and enters a new OBP alias definition for the new disk:

```
# prtvtoc -h /dev/rdsk/c0t4d0s2
       0      2    00    2110976  16812416  18923391
       2      5    00          0  35368272  35368271
       3     15    01          0      9424      9423
       4     14    01       9424  35358848  35368271
# eeprom nvramrc
devalias vx-osdg01 /ssm@0,0/pci@18,700000/pci@1/scsi@2/disk@2,0:a
devalias vx-osdg02 /ssm@0,0/pci@18,700000/pci@1/scsi@2/disk@3,0:a
devalias vx-osdg03 /ssm@0,0/pci@18,700000/pci@1/scsi@2/disk@4,0:a
```

5. We still need partitions over the remaining OS volumes and, in case of a damaged boot block within the **rootvol**, the **installboot** command to recreate it. These tasks are best performed by the **vxbootsetup** script. A hint for deep-sea divers: VxVM also provides some interesting low-level commands executed by **vxbootsetup**: **vxpartadd**, **vxedvtoc**, **vxmksdpart**, and **vxeeprom**.

```
# vxbootsetup -g osdg osdg03
# prtvtoc -h /dev/rdsk/c0t4d0s2
       0      2    00    2110976  16812416  18923391
       1      3    01       9424   2101552   2110975
       2      5    00          0  35368272  35368271
       3     15    01          0      9424      9423
       4     14    01       9424  35358848  35368271
       5      0    00   25218624   8392072  33610695
       6      7    00   18923392   6295232  25218623
```

6. The OBP attribute **boot-device** should be extended by an entry for the new OS disk:

```
# eeprom boot-device
boot-device=vx-osdg01 vx-osdg02 disk net
# eeprom boot-device='vx-osdg01 vx-osdg02 vx-osdg03 disk net'
# eeprom boot-device
boot-device=vx-osdg01 vx-osdg02 vx-osdg03 disk net
```

Note: Whenever issuing a **vxbootsetup** command on an OS disk, make sure beforehand, **rootvol** already furnished with partition 0 by the **vxassist mirror** command still keeps this partition association, and partition 1 designed to become the swap partition is still unused. Otherwise the **vxbootsetup** command will fail.

7. Our preparatory tasks created a third mirror still as a current and active part of the OS volumes. Inadvertent commands would remove, destroy or invalidate data on all three mirrors. We need a frozen copy of the current data, but nevertheless capable to boot from and to recover the volume to the frozen data set. We already discussed thoroughly several techniques provided by VxVM to establish a snapshot of a volume in chapter 9. The current chapter demonstrates another way to handle a bootable snapshot based on a procedure specific to OS volumes: They provide two independent drivers accessing the same data set, the volume and the partition driver.

In order to minimize plex operations, our procedure simply offlines the plexes holding the frozen data set.

```
# vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'
opt-03
rootvol-03
swapvol-03
var-03
# vxmend -g osdg off $(vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"')
# vxprint -g osdg -pthe 'pl_sd.sd_dmname="osdg03"'
PL NAME         VOLUME       KSTATE   STATE    LENGTH   LAYOUT    NCOL/WID MODE
SD NAME         PLEX         DISK     DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE
SV NAME         PLEX         VOLNAME  NVOLLAYR LENGTH   [COL/]OFF AM/NM    MODE
SC NAME         PLEX         CACHE    DISKOFFS LENGTH   [COL/]OFF DEVICE   MODE

pl opt-03       opt          DISABLED OFFLINE  8392072  CONCAT    -        RW
sd osdg03-04    opt-03       osdg03   25209200 8392072  0         c0t4d0   ENA

pl rootvol-03   rootvol      DISABLED OFFLINE  16812416 CONCAT    -        RW
```

```
sd osdg03-02    rootvol-03   osdg03   2101552  16812416 0        c0t4d0   ENA

pl swapvol-03   swapvol      DISABLED OFFLINE  2101552  CONCAT   -        RW
sd osdg03-01    swapvol-03   osdg03   0        2101552  0        c0t4d0   ENA

pl var-03       var          DISABLED OFFLINE  6295232  CONCAT   -        RW
sd osdg03-03    var-03       osdg03   18913968 6295232  0        c0t4d0   ENA
```

8. But how is it possible to boot from the emergency disk which provides the frozen copies of the OS volumes? We already implemented some of the prerequisites by invoking **vxbootsetup** (step 5 above): We put partitions over the subdisks, we re-installed as a fallback the boot block on the root partition, and we created an OBP alias. Obviously, it is useless to perform a VxVM based boot, for the plexes determining the frozen data set are offlined. But we still have the partition drivers to the emergency disk, obstinately and unteachable ignoring any plex states. Booting from the emergency disk exclusively by the partition drivers require a modified kernel configuration file **/etc/system** to skip root volume mapping on the root partition during kernel initialization and, what is more, **/etc/vfstab** containing partition, not volume drivers.

Our first step will map the subdisks to their corresponding partition numbers in order to create the correct **/etc/vfstab**. The basics and the procedures were already explained in the "Full Battleship" part beginning on page 352, so we simply refer to them. Our file **/etc/vfstab** for the emergency disk will contain the following entries instead of the volume drivers:

```
/dev/dsk/c0t4d0s0   /dev/rdsk/c0t4d0s0   /      ufs    1    no     logging
/dev/dsk/c0t4d0s1   -                    -      swap   -    no     -
/dev/dsk/c0t4d0s5   /dev/rdsk/c0t4d0s5   /opt   ufs    1    yes    logging
/dev/dsk/c0t4d0s6   /dev/rdsk/c0t4d0s6   /var   ufs    1    no     logging
```

9. We offlined the emergency plexes while the OS was running, naturally. A file system check is required before mounting them. Then, we are able to modify **/etc/system** and **/etc/vfstab** only on the emergency partition. As described in the section about booting without VxVM beginning on page 406, we copy the versions of these files which allow a partition boot to the active location.

```
# fsck -y /dev/rdsk/c0t4d0s0
** /dev/rdsk/c0t4d0s0
** Last Mounted on /
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3a - Check Connectivity
** Phase 3b - Verify Shadows/ACLs
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cylinder Groups
211337 files, 5712887 used, 2541378 free (92722 frags, 306082 blocks, 1.1% frag-
mentation)
# fsck -y /dev/rdsk/c0t4d0s5
```

```
** /dev/rdsk/c0t4d0s5
** Last Mounted on /opt
[…]
# fsck -y /dev/rdsk/c0t4d0s6
** /dev/rdsk/c0t4d0s6
** Last Mounted on /var
[…]
# mount /dev/dsk/c0t4d0s0 /mnt
# cd /mnt/etc
# cp system.part system
# cp vfstab.c0t4d0s2 vfstab
# cd /
# umount /mnt
```

10. Assume the current OS does not work anymore or has lost critical files. Therefore, we must boot from the emergency disk based on partitions. Nevertheless, the system startup procedure will start VxVM daemons as well. Quite correct, for we want to synchronize the damaged parts of the OS volumes from the offlined emergency plexes! Since our system afterwards should run on volumes, not on partitions, we reboot just to the single user mode/milestone based on partitions to swap the role of the data sets. The current section uses **/sbin/sh** syntax in order to conform to boot process recommendations.

```
# reboot vx-osdg03 -s
[…]
Rebooting with command: boot vx-osdg03 -s
[…]
Booting to milestone "milestone/single-user:default".
[…]
VxVM sysboot INFO V-5-2-3409 starting in boot mode...
[…]
VxVM sysboot INFO V-5-2-3390 Starting restore daemon...
Requesting System Maintenance Mode
SINGLE USER MODE

Root password for system maintenance (control-d to bypass): password
# df -k / /var
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/dsk/c0t4d0s0    8262473 5729317 2450532    71%    /
/dev/dsk/c0t4d0s6    3099238 1269603 1767651    42%    /var
# vxprint -rtg osdg rootvol
[…]
v  rootvol      -            ENABLED  ACTIVE   16812416 ROUND     -        root
pl rootvol-01   rootvol      ENABLED  ACTIVE   16812416 CONCAT    -        RW
sd osdg01-02    rootvol-01   osdg01   2101551  16812416 0         c0t2d0   ENA
pl rootvol-02   rootvol      ENABLED  ACTIVE   16812416 CONCAT    -        RW
sd osdg02-02    rootvol-02   osdg02   2101552  16812416 0         c0t3d0   ENA
pl rootvol-03   rootvol      DISABLED OFFLINE  16812416 CONCAT    -        RW
```

```
sd osdg03-02    rootvol-03   osdg03   2101552  16812416 0         c0t4d0   ENA
# vxvol -g osdg stopall
# vxprint -rtg osdg rootvol
[…]
v  rootvol      -            DISABLED CLEAN    16812416 ROUND     -        root
pl rootvol-01   rootvol      DISABLED CLEAN    16812416 CONCAT    -        RW
sd osdg01-02    rootvol-01   osdg01   2101551  16812416 0         c0t2d0   ENA
pl rootvol-02   rootvol      DISABLED CLEAN    16812416 CONCAT    -        RW
sd osdg02-02    rootvol-02   osdg02   2101552  16812416 0         c0t3d0   ENA
pl rootvol-03   rootvol      DISABLED OFFLINE  16812416 CONCAT    -        RW
sd osdg03-02    rootvol-03   osdg03   2101552  16812416 0         c0t4d0   ENA
# vxprint -g osdg -pne 'pl_sd.sd_dmname~/^osdg0[12]$/'
opt-01
opt-02
rootvol-01
rootvol-02
swapvol-01
swapvol-02
var-01
var-02
# vxmend -g osdg -o force off `
  vxprint -g osdg -pne 'pl_sd.sd_dmname~/^osdg0[12]$/'`
# vxmend -g osdg on `
  vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'`
# vxprint -rtg osdg rootvol
[…]
v  rootvol      -            DISABLED CLEAN    16812416 ROUND     -        root
pl rootvol-01   rootvol      DISABLED OFFLINE  16812416 CONCAT    -        RW
sd osdg01-02    rootvol-01   osdg01   2101551  16812416 0         c0t2d0   ENA
pl rootvol-02   rootvol      DISABLED OFFLINE  16812416 CONCAT    -        RW
sd osdg02-02    rootvol-02   osdg02   2101552  16812416 0         c0t3d0   ENA
pl rootvol-03   rootvol      DISABLED STALE    16812416 CONCAT    -        RW
sd osdg03-02    rootvol-03   osdg03   2101552  16812416 0         c0t4d0   ENA
# vxmend -g osdg fix clean `
  vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'`
# vxprint -rtg osdg rootvol
[…]
v  rootvol      -            DISABLED CLEAN    16812416 ROUND     -        root
pl rootvol-01   rootvol      DISABLED OFFLINE  16812416 CONCAT    -        RW
sd osdg01-02    rootvol-01   osdg01   2101551  16812416 0         c0t2d0   ENA
pl rootvol-02   rootvol      DISABLED OFFLINE  16812416 CONCAT    -        RW
sd osdg02-02    rootvol-02   osdg02   2101552  16812416 0         c0t3d0   ENA
pl rootvol-03   rootvol      DISABLED CLEAN    16812416 CONCAT    -        RW
sd osdg03-02    rootvol-03   osdg03   2101552  16812416 0         c0t4d0   ENA
# cd /etc
# cp vfstab.vol vfstab
# cp system.vol system
# reboot vx-osdg03
```

11. The system boots once again from the emergency disk, but based on volumes this time. Maybe you need to restore some files from the still not resynchronized damaged boot disks (that's why we kept their OFFLINE state). Simply use the partition drivers to access them. Then, you should synchronize the volumes to achieve redundancy. Counting four volumes, we expect four synchronization threads, sequentially executed. Surprisingly, **vxtask list** (without disk group specification) lists eight tasks. An extract of the disk group configuration during synchronization (in our example to the **opt** volume) indeed shows, that each stale plex is resynchronized by a particular thread — for whatever reason.

```
# fsck -y /dev/rdsk/c0t2d0s0
** /dev/rdsk/c0t2d0s0
** Last Mounted on /
[…]
# mount /dev/dsk/c0t2d0s0 /mnt
# cp /mnt/<path>/<file> /<path>
# umount /mnt
# vxmend -g osdg on `
  vxprint -g osdg -pne 'pl_sd.sd_dmname~/^osdg0[12]$/'`
# vxprint -rtg osdg rootvol
v  rootvol      -            ENABLED  ACTIVE   16812416 ROUND     -        root
pl rootvol-01   rootvol      DISABLED STALE    16812416 CONCAT    -        RW
sd osdg01-02    rootvol-01   osdg01   2101551  16812416 0         c0t2d0   ENA
pl rootvol-02   rootvol      DISABLED STALE    16812416 CONCAT    -        RW
sd osdg02-02    rootvol-02   osdg02   2101552  16812416 0         c0t3d0   ENA
pl rootvol-03   rootvol      ENABLED  ACTIVE   16812416 CONCAT    -        RW
sd osdg03-02    rootvol-03   osdg03   2101552  16812416 0         c0t4d0   ENA
# vxrecover -g osdg -b
# vxtask list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
VxVM vxtask WARNING V-5-1-2497 Unable to get disk group record: Record not in
disk group
   161           PARENT/R  0.00% 8/0(1) VXRECOVER 539720.0
   162   162     ATCOPY/R 03.90% 0/8392072/327680 PLXATT 0.1052 0.1051 0.1
# vxtask -g osdg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   162   162     ATCOPY/R 05.76% 0/8392072/483328 PLXATT opt opt-01 osdg
# vxprint -rtg osdg opt
v  opt          -            ENABLED  ACTIVE   8392072  ROUND     -        fsgen
pl opt-01       opt          ENABLED  STALE    8392072  CONCAT    -        WO
sd osdg01-04    opt-01       osdg01   18913967 8392072  0         c0t2d0   ENA
pl opt-02       opt          DISABLED STALE    8392072  CONCAT    -        RW
sd osdg02-03    opt-02       osdg02   18913968 8392072  0         c0t3d0   ENA
pl opt-03       opt          ENABLED  ACTIVE   8392072  CONCAT    -        RW
sd osdg03-04    opt-03       osdg03   25209200 8392072  0         c0t4d0   ENA
```

12. As the final step, the plexes containing subdisks on the emergency disk should be offlined, file system checked, and prepared for the next case of emergency — hopefully not too soon. The gap is closed.

```
# vxmend -g osdg off `vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'`
# fsck -y /dev/rdsk/c0t4d0s0
# fsck -y /dev/rdsk/c0t4d0s5
# fsck -y /dev/rdsk/c0t4d0s6
# mount /dev/dsk/c0t4d0s0 /mnt
# cd /mnt/etc
# cp system.part system
# cp vfstab.c0t4d0s2 vfstab
# cd /
# umount /mnt
```

You will surely agree, that we need to execute a lot of complicated commands. Why not write Shell scripts to make life easier and even to automate recovery? We present three scripts, the first called **EmergencyDisk_Prepare** to prepare the emergency disk (steps 11 and 12/7, 8 and 9), the second (**EmergencyDisk_Restore**) to boot the system from the emergency disk based on volumes in order to recover them (step 10), and finally a run level script (**NOT_S99EmergencyDisk_Restore**) to automate the latter task, typically disabled, but temporarily enabled by the first script.

Script **EmergencyDisk_Prepare**

```
#!/sbin/sh
echo Recovering OS disks. Please wait ...
vxmend -g osdg on `vxprint -g osdg -pne 'pl_sd.sd_dmname~/^osdg0[12]$/'`
vxrecover -g osdg || exit 1
echo FS checking partitions of emergency disk c0t4d0. Please wait ...
vxmend -g osdg off `vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'`
fsck -y /dev/rdsk/c0t4d0s0
fsck -y /dev/rdsk/c0t4d0s5
fsck -y /dev/rdsk/c0t4d0s6
mount /dev/dsk/c0t4d0s0 /mnt
cd /mnt/etc
cp system.part system
cp vfstab.c0t4d0s2 vfstab
cp rcS.d/NOT_S99EmergencyDisk_Restore rcS.d/S99EmergencyDisk_Restore
cd /
umount /mnt
echo The disk c0t4d0 is ready for emergency use.
exit 0
```

Script **EmergencyDisk_Restore**

```
#!/sbin/sh
vxvol -g osdg stopall
vxmend -g osdg -o force off `
vxprint -g osdg -pne 'pl_sd.sd_dmname~/^osdg0[12]$/'`
vxmend -g osdg on `vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'`
vxmend -g osdg fix clean `vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'`
cd /etc
cp vfstab.vol vfstab
cp system.vol system
mv rcS.d/S99EmergencyDisk_Restore rcS.d/NOT_S99EmergencyDisk_Restore
reboot vx-osdg03
```

Script **NOT_S99EmergencyDisk_Restore**

```
#!/sbin/sh
echo Stopping all volumes of disk group osdg
vxvol -g osdg stopall || exit 1
echo Offlining all plexes of disks osdg01, osdg02 ...
vxmend -g osdg -o force off `
vxprint -g osdg -pne 'pl_sd.sd_dmname~/^osdg0[12]$/'` || exit 2
echo Onlining all plexes of emergency disk osdg03 ...
vxmend -g osdg on `vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'` || exit 2
vxmend -g osdg fix clean `vxprint -g osdg -pne 'pl_sd.sd_dmname="osdg03"'` ||
exit 2
echo Creating /etc/vfstab and /etc/system for encapsulated OS disks
cd /etc
cp vfstab.vol vfstab
cp system.vol system
mv rcS.d/S99EmergencyDisk_Restore rcS.d/NOT_S99EmergencyDisk_Restore
echo Rebooting using encapsulated disk osdg03
reboot vx-osdg03
```

11.8 Hot Relocation Troubles

11.8.1 Plex Synchronization Skipped

In all software versions of VxVM we know, except for the unpatched version of VxVM 5.0, **vxrelocd** shows a strange failure handling under specific conditions. Within a diskgroup containing three disks we create three volumes: **vol1** unmirrored, **vol2** holding two, **vol3** three data plexes. All volumes use a subdisk on the common disk **adg01**. We will make this disk unavailable to VxVM in order to analyze the failure handling of **vxrelocd**.

```
# vxassist -g adg make vol3 1g layout=mirror nmirror=3 init=active
# vxassist -g adg make vol2 1g layout=mirror nmirror=2 init=active \
  alloc=adg01,adg02
# vxassist -g adg make vol1 1g alloc=adg01
# vxprint -rtg adg
[…]
dm adg01        c1t1d0s2     auto     2048     122171136 -
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 -

v  vol1         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01      vol1         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-03     vol1-01      adg01    4194304  2097152  0         c1t1d0   ENA

v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-02     vol2-01      adg01    2097152  2097152  0         c1t1d0   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-02     vol2-02      adg02    2097152  2097152  0         c1t1d1   ENA

v  vol3         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol3-01      vol3         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg01-01     vol3-01      adg01    0        2097152  0         c1t1d0   ENA
pl vol3-02      vol3         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01     vol3-02      adg02    0        2097152  0         c1t1d1   ENA
pl vol3-03      vol3         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg03-01     vol3-03      adg03    0        2097152  0         c1t1d2   ENA
# dd if=/dev/zero of=/dev/rdsk/c1t1d0s2 bs=128k oseek=1 count=1
# vxconfigd -k
```

From a previous section of this chapter, we already know the default behavior of VxVM and especially of **vxrelocd**: All plexes affected by failed subdisks on the failed disk enter kernel state DISABLED and application state NODEVICE. As long as there are still other healthy plexes within a volume, the volume keeps its state pair ENABLED/ACTIVE, thus providing continued access. After some seconds, **vxrelocd** searches for subdisk replacement. A DISABLED volume (**vol1** in our example) cannot relocate the failed subdisk, because it has no valid plex anymore to synchronize from. It is therefore skipped by **vxrelocd**. An ENABLED volume (**vol2**) will get a replacement subdisk (step 1) to become a synchronized member of the volume (step 2). But what happens, if **vxrelocd** cannot find the required space to relocate the failed subdisk (**vol3**, because we only have three disks within the disk group)? Well, the answer is quite simple: No subdisk relocation is performed.

Unfortunately, **vxrelocd** does not behave like this. Look at the following result created by **vxrelocd** in our example.

```
# vxprint -rtg adg
[…]
dm adg01        -            -        -        -        NODEVICE
dm adg02        c1t1d1s2     auto     2048     122171136 -
dm adg03        c1t1d2s2     auto     2048     122171136 -
```

```
v  vol1        -          DISABLED ACTIVE   2097152  SELECT    -        fsgen
pl vol1-01     vol1       DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg01-03    vol1-01    adg01    4194304  2097152  0         -        NDEV

v  vol2        -          ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01     vol2       DISABLED IOFAIL   2097152  CONCAT    -        RW
sd adg03-02    vol2-01    adg03    2097152  2097152  0         c1t1d2   ENA
pl vol2-02     vol2       ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-02    vol2-02    adg02    2097152  2097152  0         c1t1d1   ENA

v  vol3        -          ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol3-01     vol3       DISABLED NODEVICE 2097152  CONCAT    -        RW
sd adg01-01    vol3-01    adg01    0        2097152  0         -        NDEV
pl vol3-02     vol3       ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-01    vol3-02    adg02    0        2097152  0         c1t1d1   ENA
pl vol3-03     vol3       ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg03-01    vol3-03    adg03    0        2097152  0         c1t1d2   ENA
```

vol1 is indeed DISABLED, **vol3** could not find another disk for subdisk relocation, so the affected plex remained in state DISABLED/NODEVICE. Although **vol2**, as expected, got the subdisk relocated to the third disk (**adg03**), **vxrelocd** skipped the plex synchronization, leaving the plex in the DISABLED/IOFAIL state (DISABLED/RECOVER in case of a complete disk failure).

The mails sent by **vxrelocd** give no further hints on the reason for the incomplete subdisk relocation. Some excerpts:

```
Subject: Volume Manager failures on host haensel
[…]
Failures have been detected by the VERITAS Volume Manager:

failed disks:
 adg01

failed plexes:
vol1-01
vol2-01
vol3-01

The Volume Manager will attempt to find spare disks, relocate failed
subdisks and then recover the data in the failed plexes.

Subject: Volume Manager failures on host haensel
[…]
Attempting to relocate subdisk adg01-02 from plex vol2-01.
Dev_offset - length 2097152 dm_name adg01 da_name c1t1d0s2.
The available plex vol2-02 will be used recover the data.
```

```
Subject: Volume Manager failures on host haensel
[…]
Attempting to relocate subdisk adg01-01 from plex vol3-01.
Dev_offset - length 2097152 dm_name adg01 da_name c1t1d0s2.
The available plex vol3-02 vol3-03 will be used recover the data.

[…]
Subject: Attempting VxVM relocation on host haensel
[…]
Relocation was not successful for subdisks on disk adg01 in
volume vol2 in disk group adg.  No replacement was made and the
disk is still unusable.
[…]
Subject: Attempting VxVM relocation on host haensel
[…]
Volume vol2 Subdisk adg01-02 relocated to adg03-02,
but not yet recovered.
[…]
Subject: Attempting VxVM relocation on host haensel
[…]
Relocation was not successful for subdisks on disk adg01 in
volume vol3 in disk group adg.  No replacement was made and the
disk is still unusable.
[…]
```

Due to a misplaced break condition in a loop, the recovery synchronization of volume **vol2** is not started, for **vol3** (!) could not find a subdisk replacement. In order to complete the recovery procedure, just execute the appropriate recovery command manually:

```
# vxrecover -g adg -b vol2
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   164   164     ATCOPY/R 32.23% 0/2097152/675840 PLXATT vol2 vol2-01 adg
# vxprint -rtg adg vol2
[…]
v  vol2         -            ENABLED  ACTIVE   2097152  SELECT    -        fsgen
pl vol2-01      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg03-02     vol2-01      adg03    2097152  2097152  0         c1t1d2   ENA
pl vol2-02      vol2         ENABLED  ACTIVE   2097152  CONCAT    -        RW
sd adg02-02     vol2-02      adg02    2097152  2097152  0         c1t1d1   ENA
```

11.8.2 Unrelocation of Split Subdisks

Hot relocation may not always find appropriate disk space to execute a one-to-one subdisk replacement. Will the hot relocation procedure fail, if a failed subdisk cannot be recreated on one target disk? Or will the subdisk be split into smaller fragments spread over multiple disks? Since this is a somewhat realistic scenario, we should examine VxVM's behavior. Within a disk group containing four disks at approx. 17 GB in size, we create a mirrored volume of 16 GB on the first two disks, while a mirrored dummy volume of 8 GB covers the last two disks. So, if one disk of the regular volume fails, the subdisk on it cannot be replaced by just one subdisk.

```
# vxassist -g adg make vol 16g layout=mirror alloc=adg01,adg02 init=active
# vxassist -g adg make dummy 8g layout=mirror alloc=adg03,adg04 init=active
# vxprint -rtg adg
[…]
dm adg01        c4t1d0s2     auto     2048     35365968 -
dm adg02        c4t2d0s2     auto     2048     35365968 -
dm adg03        c4t3d0s2     auto     2048     35365968 -
dm adg04        c4t4d0s2     auto     2048     35365968 -

v  dummy        -            ENABLED  ACTIVE   16777216 SELECT    -        fsgen
pl dummy-01     dummy        ENABLED  ACTIVE   16777216 CONCAT    -        RW
sd adg03-01     dummy-01     adg03    0        16777216 0         c4t3d0   ENA
pl dummy-02     dummy        ENABLED  ACTIVE   16777216 CONCAT    -        RW
sd adg04-01     dummy-02     adg04    0        16777216 0         c4t4d0   ENA

v  vol          -            ENABLED  ACTIVE   33554432 SELECT    -        fsgen
pl vol-01       vol          ENABLED  ACTIVE   33554432 CONCAT    -        RW
sd adg01-01     vol-01       adg01    0        33554432 0         c4t1d0   ENA
pl vol-02       vol          ENABLED  ACTIVE   33554432 CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        33554432 0         c4t2d0   ENA
# vxdg -g adg free
DISK         DEVICE       TAG          OFFSET     LENGTH     FLAGS
adg01        c4t1d0s2     c4t1d0       33554432   1811536    -
adg02        c4t2d0s2     c4t2d0       33554432   1811536    -
adg03        c4t3d0s2     c4t3d0       16777216   18588752   -
adg04        c4t4d0s2     c4t4d0       16777216   18588752   -
```

Disk **adg01** is powered off and some I/O issued on the regular volume. Hot relocation detects the need for subdisk relocation. We turned on Shell debugging within the **vxrelocd** script to analyze its behavior:

```
+ vxassist -r -g adg move vol !adg01 adg02
+ vxassist -r -g adg move vol !adg01 adg03
+ vxassist -r -g adg move vol !adg01 adg04
+ vxassist -r -g adg move vol spare=yes !adg01
```

The script **vxrelocd** tried to find an appropriate disk as subdisk container by executing **vxassist move** commands. The first command failed, because disk **adg02** was already in use by the other volume mirror. The next two commands failed, because disks **adg03** and **adg04** each could not provide the required free space. The fourth command succeeded, for only disk **adg01** was excluded from the storage allocation, thus enabling multiple target disks. **spare=yes** assigned a higher priority to spare disks (which we did not define), the option **-r** included spare disks in the space calculation for the subdisk relocation.

Indeed, the failed subdisk is relocated to two split subdisks. That is good news. These subdisks are synchronized in the known way:

```
# vxprint -rtg adg vol
[…]
v  vol          -            ENABLED  ACTIVE   33554432 SELECT    -        fsgen
pl vol-01       vol          ENABLED  STALE    33554432 CONCAT    -        WO
sd adg03-02     vol-01       adg03    16777216 18588752 0         c4t3d0   ENA
sd adg04-02     vol-01       adg04    16777216 14965680 18588752  c4t4d0   ENA
pl vol-02       vol          ENABLED  ACTIVE   33554432 CONCAT    -        RW
sd adg02-01     vol-02       adg02    0        33554432 0         c4t2d0   ENA
# vxtask -g adg list
TASKID  PTID TYPE/STATE    PCT   PROGRESS
   165   165     ATCOPY/R 07.15% 0/33554432/2398208 PLXATT vol vol-01 adg
```

Well, so far so good! But what about those subdisk attributes of the two relocated subdisks storing the original disk media name and the original subdisk offset?

```
# vxprint -g adg -sF '%name %orig_dmname %orig_dmoffset' -e \
  'sd_orig_dmname!=""'
adg03-02 adg01 0
adg04-02 adg01 18588752
```

That looks quite encouraging! The correct offsets were calculated and stored. We expect a proper unrelocation with just one harmless disadvantage: Both subdisks will be unrelocated, but not joined into one subdisk. In order to analyze the procedure, we turn on Shell debugging once again.

```
# vxunreloc -g adg adg01
```

The interesting part of the debugging output together with the content of the subdisk creation template file:

```
+ vxmake -g adg -d /tmp/ur-mksd2869
+ vxsd -g adg -o rm -d mv adg03-02 adg01-UR-001
+ vxsd -g adg -o rm -d mv adg04-02 adg01-UR-002
# cat /tmp/ur-mksd2869
sd adg01-UR-001 disk=adg01 offset=0 len=18588752 comment=UNRELOC
sd adg01-UR-002 disk=adg01 offset=18588752 len=14965680 comment=UNRELOC
```

OK, great! We still expect two separated subdisks on the original disk, but no! **vxunreloc** was able to determine, that both subdisks being physically contiguous are also contiguous within the plex. They are automatically joined. Just the name of the original subdisk is lost.

11.9 Plex States Overview

It is sometimes useful to know what all the plex states mean that are output from **vxprint** or **vxinfo.** Here is a list of plex states and their meaning, taken from the man page:

ACTIVE	Either the volume is started and the plex is enabled, or the volume was not stopped cleanly and the plex was valid when the volume was stopped.
CLEAN	The plex contains valid data and the volume was stopped cleanly.
DCOSNP	A data change object (DCO) plex that is attached to a volume, and which can be used by a snapshot plex to create a DCO volume during a snapshot operation.
EMPTY	The plex is part of a volume that has not yet been initialized.
IOFAIL	The plex was detached because of an uncorrectable I/O failure on one of the subdisks in the plex.
LOG	A dirty region logging (DRL) or RAID-5 log plex.
NODAREC	No physical disk was found for one of the subdisks in the plex. This implies either that the physical disk failed, making it unrecognizable, or that the physical disk is no longer attached through a known access path.
NODEVICE	A physical device could not be found corresponding to the disk ID in the disk media record for one of the subdisks associated with the plex. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.
OFFLINE	The plex was disabled using the vxmend off operation.
REMOVED	A physical disk used by one of the subdisks in the plex was removed through administrative action with vxdg -k rmdisk.
SNAPATT	The plex is being attached as part of a backup operation by the vxassist snapstart operation. When the attach is complete, the condition changes to SNAPDONE. A system reboot or manual starting of the volume removes the plex and all of its subdisks.

SNAPDIS	A vxassist snapstart operation completed the process of attaching the plex. It is a candidate for selection by the vxplex snapshot operation. A system reboot or manual starting of the volume dissociates the plex.
SNAPDONE	A vxassist snapstart operation completed the process of attaching the plex. It is a candidate for selection by the vxassist snapshot operation. A system reboot or manual starting of the volume removes the plex and all of its subdisks.
SNAPTMP	The plex is being attached as part of a backup operation by the vxplex snapstart operation. When the attach is complete, the condition changes to SNAPDIS. A system reboot or manual starting of the volume dissociates the plex.
STALE	The plex does not contain valid data, either as a result of a disk replacement affecting one of the subdisks in the plex, or as a result of an administrative action on the plex such as vxplex det.
TEMP	The plex is associated temporarily as part of a current operation, such as vxplex cp or vxplex att. A system reboot or manual starting of a volume dissociates the plex.
TEMPRM	The plex was created for temporary use by a current operation. A system reboot or manual starting of a volume removes the plex.
TEMPRMSD	The plex and its subdisks were created for temporary use by a current operation. A system reboot or manual starting of the volume removes the plex and all of its subdisks.