DEC OSF/1 Logical Storage Manager

CookBook

Wai C. Yim
DEC OSF/1 Logical Storage

Modification History

Version	Date	Who	Reason
00.00	**02/24/94**	**yim**	**Document creation**
00.10	02/24/94	yim	Completed Outline
00.11	03/01/94	yim	Completed Introduction Chapter
00.12	03/01/94	yim	Completed Product Information Chapter
00.13	03/03/94	yim	Completed LSM Concepts Chapter
00.14	03/03/94	yim	Completed Configuring LSM Chapter
00.15	03/09/94	yim	Completed System Administration Commands Chapter
	03/22/94	yim	Modified System Administration Commands Chapter
00.16	03/22/94	yim	Completed Configuration Examples Chapter
00.17	03/24/94	yim	Completed LSM Mirroring Chapter
00.18	03/29/94	yim	Completed LSM Striping Chapter
00.19	03/29/94	yim	Completed Large Configuration Chapter
00.20	04/07/94	yim	Completed Trouble Shooting Chapter
00.21	04/07/94	yim	Completed LSM & UFS Chapter
00.22	04/08/94	yim	Completed LSM & AdvFS Chapter
00.23	04/11/94	yim	Completed LSM & RAID5 Chapter
00.24	04/11/94	yim	Completed LSM & ASE Chapter
00.50	**04/11/94**	**yim**	**Completed First Draft**
00.51	04/20/94	yim	Incorporated Cheryl Wiecek's Comments
00.52	04/20/94	yim	Incorporated Todd Kaehler's Comments
00.53	04/22/94	yim	Incorporated Maria Vella's Comments
00.54	04/27/94	yim	Incorporated Bill Dallas's Comments
00.55	04/28/94	yim	Incorporated Vasu Subramanian's Comments
00.56	04/28/94	yim	Incorporated Mark Angel's Comments
00.57	04/28/94	yim	Modified LSM & AdvFS Chapter
01.00	**04/28/94**	**yim**	**Completed Version 1**

Contents

List of Figures

1 Introduction

The DEC OSF/1 Logical Storage Manager (LSM) is a robust, feature-rich component that provides primarily concatenation, mirroring, striping, and on-line storage management. Even though the system administration interface is well documented, components such as LSM are inherently difficult to manage and their interactions with other system components complex. This CookBook is an attempt to communicate what was learned during the development cycle.

1.1 CookBook Intended Audience

This CookBook is intended for internal use by Digital personnel only. Customers should not be provided with this document under any circumstances. Readers are expected to have thorough knowledge of DEC OSF/1 system administration concepts and some knowledge of DEC OSF/1 internals especially regarding storage and file systems.

1.2 CookBook Disclaimer

All information and procedures documented in this cookbook is not officially supported. Use at your own risk. All information contained in this document is subject to change without notice.

Readers are expected to have read all of the LSM documentations, all 2,000 pages of it, from cover to cover.

1.3 CookBook Reviews

"Best bed time reading published in recent years!"

- New Hampshire Times

"Great recipes your Grandmother didn't tell you about..."

- Merrimack Post

"Home remedies that really work!"

- Spit Brook Review

1.4 Acknowledgment

Knowledge contained in this book is collected from talking to many friends and colleagues in USG. Special thanks to Cheryl Wiecek, Todd Kaehler, Maria Vella, Bill Dallas, Vasu Subramanian, and Mark Angel for taking the time to help make things right!

In additonal, many DEC OSF/1, Veritas manuals, and other internal Digital publications were referenced.

2 Product Information

2.1 Origin of LSM

Before DEC OSF/1 V2.0, Logical Volume Manager (LVM) was the default logical storage component. Due to robustness issues, only concatenation is supported. A port of Ultrix Disk Shadowing (UDS) was done to temporarily relieve the pain of not having shadowing on DEC OSF/1. A port of the Ultrix striping driver was also done for the same reason.

After extensive evaluation, LSM was chosen by the DEC OSF/1 group to be the base of our long-term logical storage solution. LSM is a port of the Volume Manager (VxVM) V1.2.1.1mp of Veritas Software. Veritas VxVM is a mature logical storage package that is supported on other Unix platforms like SCO Unix and Sun Solaris.

VxVM is under active development at Veritas Software. Advanced storage features like host based RAID5 and shared disk models are part of Veritas development plans. As these enhancements become available, they will be evaluated by the DEC OSF/1 group to determine if they fit into our long-term solution.

2.2 LSM V1.0

LSM V1.0 is targeted to ship as a layered product on DEC OSF/1 V2.0 (Sterling) Complementary Product CDROM (CPCD) Vol. 2. LSM V1.0 contains all the basic logical storage features like concatenation, mirroring, striping, GUI, etc. This layered product requires a license to operate. Please refer to the LSM SPD and release notes for detailed product information.

2.3 LSM V2.0

LSM V2.0 is targeted to ship as an integrated product on DEC OSF/1 V3.0 (Gold). This shipment is primarily the same as V1.0 except for two things. LSM V2.0 has full SMP support, and concatenation is separated out as LSM-lite and no longer requires a LSM license to operate. As an integrated product, LSM is shipped with the base OS kit and can be installed as part of the base OS installation. LSM-lite is supported with the base OS license. All additional features requires a LSM license to operate.

2.4 Other Plans

Root and swap device support is being planned right now. It is being considered for Gold but unlikely to be included. Please send your requirements to LSM product management if you need this feature sooner.

There are on-going discussions on whether to purchase and incorporate host based RAID5 from Veritas as it becomes available.

3 LSM Concepts

This chapter gives a very brief architectural overview of LSM. It is not exhaustive. It only talks about the concepts that are fundamental to provide common ground for further discussions. Although you have read the LSM documentation from cover to cover, you may benefit by reviewing this material in a more condensed form.

3.1 Application View

Applications interface with LSM through volumes. A volume is a standard DEC OSF/1 special device. It has a block device interface and a character device interface. These interfaces support open, close, read, write, and ioctl calls. A volume can be used by applications in place of a traditional physical device partition. A typical application of LSM can be a file system or a database.

All block special device files of the volumes reside in the /dev/vol directory tree. All character special device files reside in the /dev/rvol directory. A disk group name is an optional part of the path name.

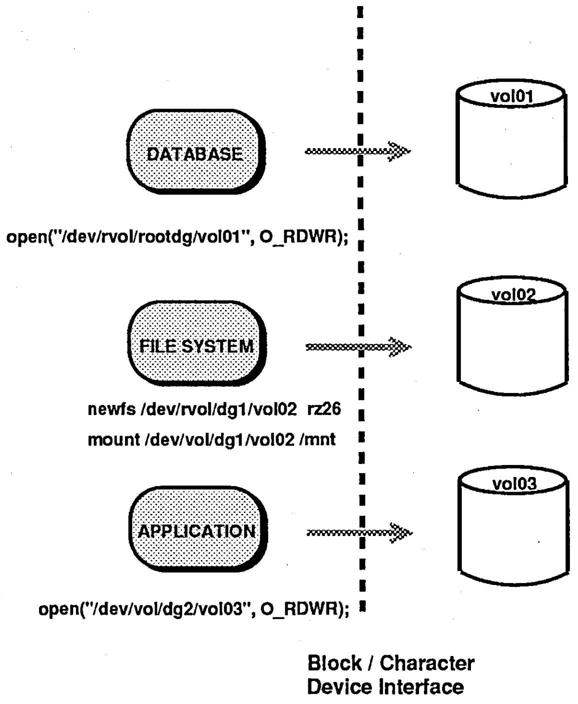

DATABASE
vol01
open("/dev/rvol/rootdg/vol01", O_RDWR);
FILE SYSTEM
vol02
newfs /dev/rvol/dg1/vol02 rz26
mount /dev/vol/dg1/vol02 /mnt
APPLICATION
vol03
open("/dev/vol/dg2/vol03", O_RDWR);
Block / Character
Device Interface

Figure 1 Application View

3.2 System Administration View

3.2.1 Graphical Interface

LSM system administration is separated into a few well defined layers. At the top, there is a Motif graphical administration interface, dxlsm. This is a comprehensive interface that interprets mouse based icon operations into LSM commands. For instance, select a volume icon with a mouse and then choose a menu item to operate on the volume.

3.2.2 Menu Driven Interface

Similar to dxlsm is a menu driven administration interface, voldiskadm. This is not supported in LSM V1.0 but will be supported in a near future release. This has many of the features dxlsm has but is a character cell interface and does not require a workstation to operate.

3.2.3 Command Line Interface

The command line layer has two sets of commands. The first set is a single top down command volassist. This command takes a volume and some parameter as input and builds all the underlying objects that are necessary for the volume. Hence, the volume is considered to be built from the top down to the bottom. The second set of commands is a group of bottom up commands like volmake, volplex, etc. These commands build individual objects and allows the construction of a volume to be performed with great precision. Hence, the volume is considered to be built from the bottom up.

3.2.4 Management Functions Library

Both sets of commands call the liblsm library to communicate with the configuration daemon, vold. This library approach not only enhances the maintainability of the commands but also allows third parties to build specialized or customized commands based on the same library calls.

3.2.5 Configuration Daemon

The configuration daemon, vold, is the centerpiece of LSM. All configuration changes are centralized through vold. This includes creation and deletion of objects, changes of configuration due to system management interaction or asynchronous error events, etc. Because these changes are done by the daemon and not the kernel, the robustness of LSM is increased. vold packages the configuration change into a transaction and passes it to the volspec driver to record the actual change.

3.2.6 Volume Special Kernel Driver

Once the volspec driver receives the transaction, it records the data on the non-volatile configuration database that resides on a set of physical disks.

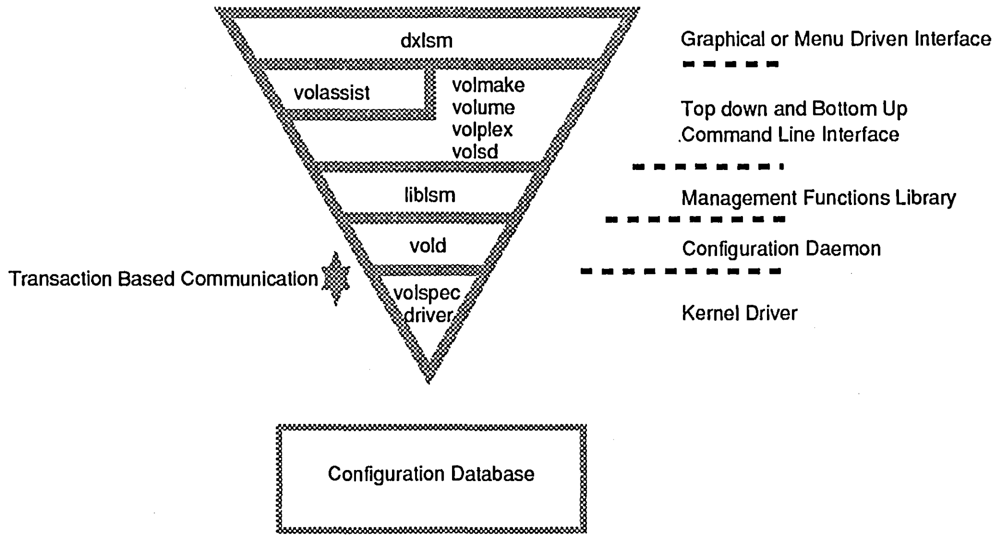

dxlsm
volassist
volmake
volume
volplex
volsd
liblsm
vold
volspec driver
Transaction Based Communication
Configuration Database
Graphical or Menu Driven Interface
Top down and Bottom Up .Command Line Interface
Management Functions Library
Configuration Daemon
Kernel Driver

Figure 2 System Admin. View

3.3 LSM Objects

There are many types of objects in a LSM configuration. The three fundamental objects that are most important to the construction of volumes are volume, plex, and subdisks. Here are the definitions of the objects.

volume : A virtual disk device that looks to applications and file systems like a regular disk partition device. Volumes present block and raw device interfaces that are compatible in their use, with disk partition devices.

plex : A copy of a volume's logical data address space, also sometimes known as a mirror. Each plex is, at least conceptually, a copy of the volume that is maintained consistently in the presence of volume I/O and reconfigurations. Plexes can have a striped or concatenated organization (layout).

subdisk : A region of storage allocated on a disk for use with a volume. Subdisks are associated to volumes through plexes. One or more subdisks are allocated to form plexes based on the plex layout: striped or concatenated.

A volume is a virtual device with an LBN range 0 to n. Mirroring is implemented at the volume layer from attaching more than one plex to a volume. Other properties of a volume are derived from the layout of the plexes contained in the volume. A volume can contain either concatenated or striped plexes. A plex is a copy of a volume and has the same LBN range 0 to n. A plex can be concatenated or striped. In a concatenated plex, LBN numbers are serially assigned to the subdisks that made up the plex. In a striped plex, LBN numbers are assigned alternately among the different subdisks in the plex.

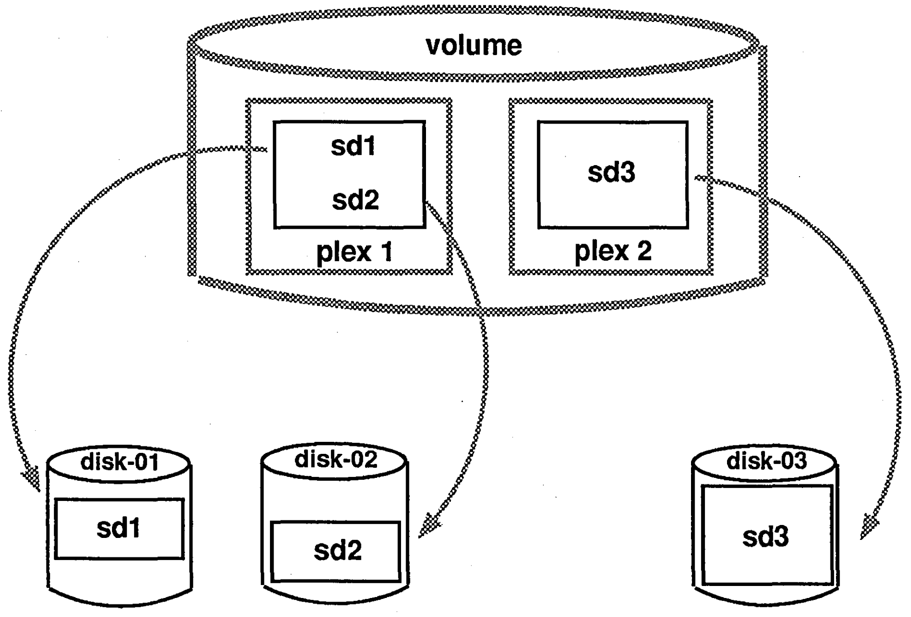


Figure 3 A Mirrored Volume

3.4 Disk Group

Disks group is an important concept in understanding the management characteristics of LSM. On a single host, disk groups can be used to simplify management and reduce the size of configuration databases. For instance, on a system with many disks, usage of the disks can be partitioned into a few disks groups by function. This way, the configuration database of each disk group is smaller and the overhead incurred in configuration changes is reduced. On a multi-host environment. disk groups are the level where failover between hosts can occur because each disk group has a self-describing configuration database.

disk group : A group of disks that share a common configuration. A configuration consists of a set of records describing objects including disks, volumes, plexes, and subdisks that are associated with one particular disk group. Each disk group has an administrator-assigned name that can be used by the administrator to reference that disk group. Each disk group has an internally defined unique disk group ID, which is used to differentiate two disk groups with the same administrator-assigned name.

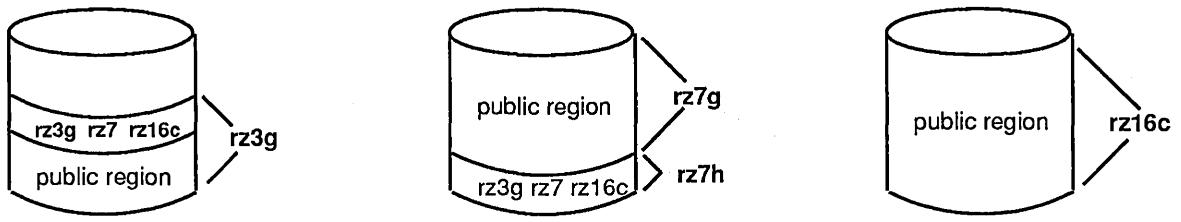
The three primary ingredients of a disk group are the collection of physical disks, a configuration database that describes physical to logical storage mappings, and a managed storage space for application data. A disk group is implemented with three kinds of LSM disks.

simple disk : LSM simple disks have both the private and public regions in the same partition.

sliced disk : LSM sliced disks use the disklabel to identify the private and public regions.

nopriv disk : LSM nopriv disks has only a public region.

The configuration database is stored on the private region of each LSM disks in a disk group except nopriv disks. The public regions of the LSM disks collectively form the storage space for application use.



LSM Simple Disk
- **of the form rz3g**
- **priv/publ region in same partition**

LSM Sliced Disk
- **of the form rz7**
- **disklabel tag for private region**
- **disklabel tag for public region**

LSM nopriv Disk
- **of the form rz16c**
- **no private region**
- **only can add to existing disk group**

Figure 4 Disk Group Implementation

4 Configuring LSM

After installing the LSM kit, some initialization is needed before LSM can be operational. This steps are documented in LSM V1.0 release notes and installation guide. Please follow the documented procedures to initialize LSM. The steps provide herein are for informational purposes only and is supplied to serve as a detailed explanation of what the documented procedures do.

4.1 Configuring LSM Manually

The rootdg disk group is the required default disk group. In addition to disk group meta data pertaining to the rootdg, it contains disk access records of LSM disks in all disk groups on the system. These disk access records are necessary for starting up any disk groups besides rootdg. Because rootdg is not enabled when the system first boots, LSM relies on a special file /etc/vol/volboot to contain the disk access records of LSM disks in rootdg. The following sections describe the steps necessary to setup rootdg and enabling LSM by hand.

1. **Starting Error Daemons**

   It is necessary to have error daemons running before starting I/Os. Otherwise, I/Os encountering errors might cause threads to hang. Generally, starting two error daemons will satisfy small configurations error handling needs. On a larger configuration, more copies of error daemons might be necessary. The command to start two copies of error daemons is as follows :

   ```
   # voliod set 2
   ```

2. **Starting *vold* in Disabled Mode**

   Because vold requires a valid configuration database to run in enabled mode, you have to start vold in disabled mode in order to set up rootdg. The command to start vold in disabled mode is as follow :

   ```
   # vold -k -m disable
   ```

3. **Initialize */etc/vol/volboot* File and Create rootdg**

   Create the special file, /etc/vol/volboot. This file contains the host ID of the host where the disk group is initialized or active. In addition, this file is used to keep track of disk access records of rootdg which are added by another command in a separate step. If no hostid is specified, the unique hostid is generated based on the IP node name of the host. A hostid can also be specified to override the generated hostid. The following command creates and initializes /etc/vol/volboot :

   ```
   # voldctl init
   ```

   or

   ```
   # voldctl init user_assigned_hostid
   ```

The required rootdg disk group needs to be created before LSM disks can be added. This is a special case of the *voldg init* command because it does not allow any LSM disk names as part of the command. The command to create rootdg is :

```
# voldg init
```

4. Initiallize a LSM Disk

A LSM disk with a private region is needed to create the rootdg. A LSM simple disk or a LSM sliced disk will serve this purpose. If you are using a LSM simple disk, make sure block 0 of the disk is not in the partition you are using. For example, if you are using partition c, issue the disklabel command to change the offset of partition c from 0 to 1, and change the size to size-1. This is needed since DEC OSF/1 does not allow writes to block 0 of a disk.

a. Initializing a simple disk

Change the disklabel :

```
# disklabel -e rz3
#           size      offset fstype   [fsize bsize cpg]
Change    c: 2050860  0      unused 1024  8192          # (Cyl. 0 - 2569)
to        c: 2050859  1      unused 1024  8192          # (Cyl. 0 - 2569)
```

Define and initialize :

```
# voldisk -f init rz3c
```

b. Initializing a sliced disk

If you are using a LSM sliced disk, issue the disklabel command to define a private region and a public region. Find two unused partition letters and define a 512 sector private and whatever size public region desired. The command to change the disklabel on rz7 is :

```
# disklabel -e rz7
#           size      offset    fstype    [fsize bsize cpg]
Change    c: 2050860  0         unused     1024  8192
          g: 819200   393216    unused     1024  8192
          h: 838444   1212416   unused     1024  819 2

to        c: 2050860  0               unused    1024 8192
          g: 2050347 1                LSMpubl   1024 8192
          h:       512 2050348        LSMpriv   1024 8192
```

The commands to initialize a LSM sliced disk on rz7 is :

```
# voldisk -f init rz7
```

5. Add the LSM Disk to rootdg and */etc/vol/volboot*

Once the LSM disk is initialized, it can be added to rootdg for use. The command to add an LSM disk to rootdg is :

```
# voldg -g rootdg adddisk rz3c
```

This command adds the rootdg LSM disk to the volboot file so rootdg can be restarted after a reboot. *vold* looks in this file to find the first disk to read the configuration database from. This disk must be a valid LSM disk in rootdg that has a private region. Therefore, it is recommended that more than one LSM disk from rootdg be listed in /etc/vol/volboot file. The command to add a LSM disk to the /etc/vol/volboot file is :

```
# voldctl add disk rz3c
```

6. **Enable *vold***

   Now you are ready to enable vold. The command to enable vold is :

```
# voldctl enable
```

At this point rootdg has completed initialization and LSM is operational.

4.2 Configuring LSM using LSM Setup Scripts

There is a set of LSM setup scriptes supplied with the LSM kit that automatically configures LSM on a newly installed system. These scripts essentially executes the same steps as described in the Manual Configruation section.

1. **volinstall**

   Immediately after installation, /sbin/volinstall can be executed to create the special device files needed for LSM daemons to communicate with the LSM kernel components. In addition, it optionally setup the system for automatic LSM restart after a reboot.

```
# volinstall
```

2. **volsetup**

   All the steps in the Manual Configuration can be replaced by executing /sbin/volsetup. The disks to be used to set up rootdg can be supplied to /sbin/volsetup. For example, if rz7 and rz3c are used to create rootdg, the command is :

```
# volsetup rz7 rz3c
```

4.3 Restarting LSM

4.3.1 Manual Restart

After rootdg is initialized, to restart LSM after a reboot is relatively simple. A restart can either be done manually or automatically. If LSM volumes are mounted in /etc/fstab, you must use the automatic restart method since the volumes must be accessible before multi-user mode. For manual restart, you need to start the error daemons, start up vold in enabled mode, and enables the volumes. The manual restart commands are :

```
# voliod set 2

# vold

# volrecover -sb
```

4.3.2 Automatic Restart

The automatic restart procedure is called lsmbstartup. This procedure essentially executes the same commands needed to start LSM manually plus a special operation to mount the system disk. This is necessary because lsmbstartup must run before /sbin/bcheckrc in order to allow volumes to be mounted in /etc/fstab. To enable automatic LSM restart, add the following line in /etc/inittab before the /sbin/bcheckrc line. Note that this can be done by /sbin/volinstall.

```
lsm:23:wait:/sbin/lsmbstartup < /dev/console > /dev/console 2>&1
fs:23:wait:/sbin/bcheckrc < /dev/console > /dev/console 2>&1
```

5 LSM System Administration Commands

There are a few layers of commands that can be used to administer a LSM configuration. Only the command line interface is discussed here. A few useful commands are described. Examples of how to use these commands can be found the the LSM Configuration Examples chapter.

5.1 Top Down Commands

volassist is the most commonly used command. It takes a volume name and an associated action to operate on the volume. For instance, you can create, mirror, grow, or shrink a volume. It has a set of defaults to allow you to specify as little information as possible. Or, you can specify arguments to override the defaults. The volassist command has the following syntax :

```
volassist [-b] [-g diskgroup] [-U usetype] [-d file] keyword arg ...
```

Refer to the man pages for details.

5.2 Bottom Up Commands

There are a few commonly used bottom up commands. volmake is used to make most objects. volume, volplex, volsd are used to manipulate volume, plex, and subdisk objects respectively. Bottom up commands allow a great deal of precision control over how objects are created and connected together. But it requires detail knowledge of LSM architecture. Here are the command syntax :

```
volmake [-U usetype] [-o useopt] [-d file] [type name [attr...]]
volume [-U usetype] [-o useopt] [-Vq] keyword arg ...
volplex [-U usetype] [-o useopt] [-V] [-v vol] keyword arg ...
volsd [-U utype] [-o uopt] [-V] [-v vol] [-p plex] keyword arg ...
```

Refer to the man pages for details.

5.3 Informational Commands

volprint is the command that displays most of the LSM configuration and status information. It is a very powerful command that has built in parsing and formatting features. It has the following syntax :

```
volprint [-AvpsdGhnlafmtqQ] [-g diskgroup] [-e pattern] [-D database] [-F [type:]format-spec]
[name ...]
```

A common form of the command that displays every object in a hierarchical one line format is :

```
# volprint -ht
```

A command to display full information on a plex vol01-01 :

```
# volprint -lp vol01-01
```

A command to display the length of a volume vol01 :

```
# volprint -v -F "%len" vol01
```

5.4 Other Useful Commands

5.4.1 Restarting volumes

volrecover is used to restart volumes and perform recovery operations for mirrored volumes. After a system reboots, volrecover can be used to enable all volumes. If a mirrored volume needs to be synchronized, volrecover starts a sync operation in the background. A command that starts all volumes and synchronizes in the background is :

```
# volrecover -sb
```

To recover a particular disk group, specify the disk group. The command to recover dg1 is :

```
# volrecover -g dg1 -sb
```

5.4.2 Removing objects

voledit is used to delete objects. One form of the command that is used frequently is to recursively remove an object and any other objects owned by it. A command to remove a volume vol01 is :

```
# voledit -rf rm vol01
```

In order to remove an object, the object cannot be owned by another object. For instance, a plex cannot be removed if it is associated with any volume. The following is an example of how to remove a plex pl1 that is associated with a volume.

```
# volplex dis pl1
# voledit -rf rm pl1
```

6 LSM Configuration Examples

6.1 Big Volume

Multiple LSM disks can be concatenated together to form a big volume. Assuming rootdg exists, you can create a concatenated volume out of the public regions available. Or you can add additional LSM disks and create volumes out of the new disks you added.

To create a 2 GB volume out of existing storage :

```
# volassist make vol01 2g
```

To create a 2GB volume with particular disks :

```
# volassist make vol01 2g disk0 disk1
```

To add LSM disks and create a 2 GB volume :

```
# voldisk -f init rz0g
# voldisk -f init rz1g
# voldisk -f init rz2g
# voldg adddisk disk0=rz0g disk1=rz1g disk2=rz2g
# volassist make vol01 2g disk0 disk1 disk2
```

To add LSM disks using voldiskadd and create a 2 GB volume :

```
# voldiskadd rz0g
# voldiskadd rz1g
# voldiskadd rz2g
# volassist make vol01 2g rz0g rz1g rz2g
```

6.2 Mirrored Volume

Multiple plexes can be associated to the same volume to create a mirrored volume. Storage of each plex should reside on different physical devices to avoid a volume being disabled by a single device failure. volassist will fail if a device that is already in the volume is specified as the mirrored plex. Only the bottom up commands can be used to mirror on the same device and it is not recommended. It is good to issue mirror commands in the background if you do not need to wait for the sync to complete before starting I/O since the sync operation can take a long time.

To add a mirror to an existing volume with any available storage :

```
# volassist mirror vol01
```

To mirror an existing volume with a particular LSM disk :

```
# volassist mirror vol01 disk0
```

To create a 3-way mirrored 1 GB volume :

```
# volassist make vol01 1g mirror=yes nmirror=3
```

To add LSM disks and create a 3-way mirrored 500 MB volume :

```
# voldisk -f init rz0g
# voldisk -f init rz1g
# voldisk -f init rz2g
# voldg adddisk disk0=rz0g disk1=rz1g disk2=rz2g
# volassist make vol01 500m mirror=yes disk0 disk1 disk2
```

6.3 Striped Volume

For higher read/write throughput, a stripe volume can be used. The basic components of a stripe volume is the size of the volume in multiples of stripe width used, stripe width, and number of stripes. Stripe blocks of the stripe width size are interleaved among the subdisks resulting in an even distribution of accesses between the subdisks. Stripe width is defaulted to 128 sectors but can be tuned to a specific application's need. The volassist command rounds down the volume length to multiples of stripe width automatically.

To create a ~3 GB 3-way stripe volume :

```
# volassist make vol01 3g layout=stripe nstripe=3
```

To create a ~ 3 GB 6-way stripe volume and stripe width to 512 sectors :

```
# volassist make vol01 3g layout=stripe nstripe=6 stwidth=512
```

To create a ~3 GB stripe volume using particular LSM disks (implied 3-way) :

```
# voldisk -f init rz0g
# voldisk -f init rz1g
# voldisk -f init rz2g
# voldg adddisk stripe_disk0=rz0g stripe_disk1=rz1g stripe_disk2=rz2g
# volassist make vol01 3g layout=stripe stripe_disk0 stripe_disk1 stripe_disk2
```

6.4 Mirrored Striped Volume

Mirrored striped volumes are used when speed and availability are both important. Note that LSM can support mirroring of stripe plexes but not striping of mirror sets. This is less than optimal in terms of failure characteristics since one failing subdisk can cause an entire striped plex to be disabled. But nonetheless, it provides sufficient availability. Note that the different striped plexes in a mirrored volume do not have to be symmetrical. For instance, a 3-way striped plex can be mirrored with a 2-way striped plex as long as the size is the same.

To add a striped mirror to an existing volume with any available storage :

```
# volassist mirror vol01 layout=stripe
```

To mirror an existing volume with particular LSM disks :

```
# volassist mirror vol01 layout=stripe disk0 disk1 disk2
```

To add LSM disks and create a 2-way mirrored 500 MB striped volume :

```
# voldisk -f init rz0g
# voldisk -f init rz1g
# voldisk -f init rz2g
# voldisk -f init rz10g
# voldisk -f init rz11g
# voldg adddisk disk0=rz0g disk1=rz1g disk2=rz2g disk10=rz10g disk11=rz11g
# volassist make vol01 500m layout=stripe nstripe=3 disk0 disk1 disk2
# volassist mirror vol01 layout=stripe disk10 disk11
```

6.5 Updating */etc/vol/volboot*

If you are making major changes to the configuration, you need to update the /etc/vol/volboot file. Remove the LSM disks entries of devices you are no longer using in the LSM configuartion and add new LSM disks entries.

```
# voldctl rm disk rz0g
# voldctl add disk rz3c
```

7 LSM Mirroring

LSM mirroring is implemented at the volume level. A volume with multiple plexes is considered mirrored. Mirroring is used mainly for availability. But there are other system administration operations that utilize the mirroring feature to operate.

Here are some discussions on general mirroring issues and some specific LSM mirroring information. This should help you better understand LSM mirroring if you are not already an expert.

7.1 Reads and Writes

The general goal of successful mirroring is to provide availability to applications transparently. In order to do this, reads and writes from a mirrored set look just like physical I/O. In the case where media problems occur on one of the mirrored plexes, recovery is provided by LSM if possible.

When a mirrored volume is in steady state, all plexes are active and contain the same data. A user read can be satisfied by reading any one and only one of the plexes. A user write must be fanned out to all plexes to keep the data consistent among them. Typically, a mirrored volume in steady state has a higher read throughput than a single spindle because all plexes can be used to satisfy user reads. The write throughput is usually slightly lower than a single spindle because all writes to all plexes must complete before the user write is considered complete.

7.2 Read Policies

If a volume is mirrored, user reads can be satisfied by any ACTIVE plex in the volume. There is a read policy associated with a mirrored volume that dictates the behavior of the reads. The read policy is determined automatically when a volume is created or when a plex is added or removed. This read policy can also be set by LSM commands. If a mirrored volume has symetrical plexes, the policy defaults to ROUND (round robin). Reads are directed to all plexes alternately. If a mirrored voume is asymetrical and has a striped plex, the policy defaults to SELECT (LSM selects the best plex). In this case, LSM assumes the striped plex has higher performance and reads are directed to the striped plex. If the policy is set to PREFER (preferred plex), reads are directed to the named plex. This policy is not set by default. You have to set it by specifying the preferred plex name. For example, if you have a RAM disk, you can set it as the preferred plex to improve your volume read throughput. An example of how to set the read policy of a volume is :

```
# volume rdpol prefer vol01 fast-plex
```

7.3 Copies and Syncs

Normally a volume is formed with one plex, the ACTIVE plex, and mirrored with a second or more TEMP plexes. A copy operation is used to copy all the data from the ACTIVE plex to the TEMP plexes. When the copy completes, the TEMP plexes are turned into ACTIVE plexes and the volume is fully mirrored. Before the copy completes, a failure on the active plex can cause the data to be unavailable. Here are the commands to create a volume with a ACTIVE plex and mirror it with a copy :

```
# volassist make vol01 1g
# volassist mirror vol01
```

A copy operation is a process driven procedure that issues atomic copy commands to a volume through the ioctl interface. An atomic copy command locks down the LBN range it is operating on to prevent user read/writes from interfering with the copy, and copies the data from the ACTIVE plex to the TEMP plex. Therefore, the copy time of a plex is roughly the time to read the data once and write the data once with a queue depth zero thread plus time lost to seeks and arbitrating for the device in the presence of user read/writes. To estimate the copy time for one temporary plex copy with no user I/Os :

(volume size * 2) / (zero queue depth disk transfer rate)

Once the copy completes, all the plexes are in ACTIVE state. All writes are fanned out to all plexes in order to keep the data consistent. If a user write completes to some plexes but not all the plexes in a mirrored volume due to system crash or another other reason, the plexes in a mirrored volume become inconsistent. This is detected by the condition where the volume is being restarted and the plexes are in ACTIVE state. A normal shutdown changes the plexes to a CLEAN. Therefore, a volume that is not enabled and contains ACTIVE plexes indicates a previous crash.

If this happens, the volume is restarted and all plexes are put into ACTIVE state and the volume is put in SYNC state. A sync operation is started to remove the inconsistency. A sync operation is similar to a copy operation. It copies data from one plex to another but all the plexes in a volume in SYNC state are considered equally valid. Even if one of the plexes fails before the sync operation completes, the data is still available from the remaining plexes.

A volume in SYNC state processes user read/writes a little differently from a volume in copy. In SYNC state, reads can be satisfied by any ACTIVE plex. But the data is written to the remaining plexes before it is returned to the user. This guarantees any inconsistency is removed so that two consecutive reads to the same LBN will not result in different data. This significantly impacts the read response time.

7.4 Management Functions utilizing Mirroring

Some LSM management functions utilize mirroring to manipulate storage. For instance, to move a subdisk, LSM adds a sparse plex with the target subdisk, copies the data from the source subdisk to the target subdisk on the sparse plex, replaces the source subdisk with the target subdisk, and removes the sparse plex. During this operation, the volume is mirrored with a sparse plex. Another instance that uses mirroring is adding a logging subdisk to a plex. The logging subdisk is added as an sparse plex before it is integrated into the associated plex.

Note that LSM-lite does not support mirroring. As a result, some LSM management functions are not supported in LSM-lite.

7.5 Creating a Volume without Copies

If an application never reads a block that is not written first, then it is safe to create a mirrored volume without first making the plexes consistent. For instance, UFS only reads regions that it has written. Not performing a copy operation saves a lot of system resources and I/O bandwidth.

The commands to create a 2-way mirrored volume and force an enable are :

```
# volmake sd disk0-01 len=1g offset=0 rz0g
# volmake sd disk1-01 len=1g offset=0 rz1g
# volmake plex vol01-01 sd=disk0-01
# volmake plex vol01-02 sd=disk1-01
# volmake -U fsgen vol vol01 plex=vol01-01,vol01-02
# volume init active vol01
```

7.6 Block Change Logging

If a system crashes while there are outstanding user writes on a mirrored volume, inconsistencies can result. Since it is not predictable where the inconsistencies are located on the volume, the entire volume needs to be synchronized. This operation has impact on system bandwidth and I/O performance. In order to avoid this expensive operation, a user can choose to log the LBN and length of each write before it goes out to the volume. This is BCL (Block Change Logging).

If BCL is enabled on a volume, all user writes must be recorded on a logging subdisk before it is issued to the volume data region. This ensures if a system crash occurs, the LBN and length of possible inconsistencies caused by outstanding writes are identifiable by the entries in the logging subdisk. When the volume is re-enabled after a reboot, the entries in the logging subdisk are used to repair the volume before it is turned into ACTIVE state.

The logging subdisk is always 1 sector in length. This is the atomic unit a digital disk device can write. To avoid the possibility of incomplete writes to the log region, the atomic transfer size is chosen to the the subdisk length.

To enable BCL, associate a logging subdisk to at least two plexes in a mirrored volume. It is recommended that you add logging subdisks to all the plexes in a mirrored volume. Any plexes that do not have a logging subdisk will not be used in the recovery process. The following commands demonstrates how to create a volume and enable BCL on it.

```
# volassist make vol01 1g mirror=yes nmirror=2
# volmake sd logsd-1 len=1 disk0
# volmake sd logsd-2 len=1 disk1
# volsd aslog vol01-01 logsd-1
# volsd aslog vol01-02 logsd-2
```

BCL adds significant overhead to the user write path. Each write has to first complete a one-block write to the logging subdisk then it is fanned out to the plexes in the volume. In a multi-threaded write application, many writes can be bundled into one log write and hence reduces the log write overhead of each I/O. In the worst case, a single threaded application doing small writes can experience response time twice as long as the non-BCL writes.

7.7 Dirty Region Logging

DRL (Dirty Region Logging) is not implemented at the moment but is being considered in a future release. DRL is an alternative logging scheme that potentially has better performance than BCL yet can achieve similar recovery characteristics. The basic idea of DRL is to divide the volume into regions of LBNs. If a user write is issued to a certain LBN, the region is marked dirty on the logging subdisk with a synchronous write. But any subsequent user writes to the same region can proceed without log write overhead because the region is already marked dirty. During a recovery, the regions marked dirty are repaired.

7.8 Failure Behavior

A mirrored volume has a different failure behavior than physical devices if media errors or device failures are encountered. It takes additional step to ensure data is available to users if possible. A simple cause of media degradation, and a case of device failure is discussed here as an example.

A user read is targeted to plex1 of a 2-way mirrored volume. A bad block is discovered at the target LBN. LSM reads the same LBN from plex2, writes to plex1 to revector the bad block, and then returns the data to user.

A user write is fanned out to plex1 and plex2 of a 2-way mirrored volume. The physical device plex2 is based on failed and cannot complete the write. The post processing routine in LSM noticed plex2 has failed and the data on it is inconsistent. It synchronously marks plex2 as IOFAIL to indicate the condition before delivering the write completion to the user. An IOFAIL condition prevents a plex from being used to satisfy any user I/Os.

8 LSM Striping

LSM striping is provided at the plex level. A striped plex maps its LBNs to alternating subdisks within the plex to spread out the I/O access evenly. This method tends to improve read/write access time of a typical I/O load. In the unusual cases like, single threaded I/O stream or multiple large I/O streams, special tuning might be necessary to obtain the desired performance improvement.

With a stripe width of n sectors, the first n LBNs of the plex is mapped to the first striping subdisk. The following n LBNs is mapped to the second striping subdisk, etc.

8.1 Reads and Writes

A read or write to a volume with a stripe plex results in a calculation of the location of the data. The calculation has two parts. 1) The starting LBN is used to locate the starting subdisk. This is done with a modula operation on the starting LBN using the stripe width. 2) Issue as many I/Os as needed to satisfy the transfer size. If the transfer size is larger than the stripe width, it might be necessary to issue I/Os to more than one stripe subdisk. To get maximum read/write performance the subdisks of a striped plex should be located on different disks and the disks should be connected to different controllers or buses if possible.

8.2 Stripe Width

The stripe width is the number of sectors that are contiguous on a subdisk before the following LBNs are mapped to the next subdisk. Stripe width is set to 128 sectors by default. It is found that this is a good number for common application loads (multiple threaded, less than 64KB, I/Os).

If an application requires large I/O transfer sizes, a balance between the transfer time and seek time must be maintained in order to take advantage of the stripe layout. In general, the goal is to spread the transfer time among the stripe subdisk spindles. The stripe width should not be too small such that the time lost to seeking on the second spindle is larger than a spiral transfer using just one spindle.

If an application is doing large spiral writes, please take into account that the vol driver breaks up I/O to 64KB internally. These I/O fragments are issued serially.

8.3 Stripe Plex Length

The length of a striped plex must be in multiples of the stripe width. If the length specified is not in multiples of stripe width during a striped plex creation, it is rounded up to the next multiple of stripe width. If you want to specify the exact size and not have volassist performance rounding, using the following formula :

**stripe plex size = (int(disk_size / stripe_width) * stripe_width * # of stripes),
where disk_size is the size of the smallest LSM disks you are using
in the stripe set.**

8.4 Stripe Set Subdisks

All striping subdisks must be the same length. In other words, you can only have a stripe plex as large as the number of devices you have times the smallest available space among the devices.

8.5 System Admin. Implications

Using striped plexes has some system admin. implications. A striped plex cannot be resized. Striping subdisks cannot be split or replaced.

9 Large Configuration

LSM defaults are set to favor a small or medium size configuration. Large configurations work correctly with the defaults but can be tuned to improve manageability. Configurations over 30 GB or 20 disks need special considerations.

9.1 Private Region Size

The default private region size is 512 sectors. It typically can contain up to 300 records. Each object created in a disk group has a record in the private region. Objects include, LSM disks, subdisks, plexes, volumes, disk groups, and LSM-disk disk access records. In addition, all LSM disks in all disk groups have a record in rootdg. You should try to anticipate the size of the configuration database and allocate an appropriate private region size up front if possible. Otherwise, all LSM disks have to be removed and reintegrated with a larger private region size when the configuration database runs out of room.

A very rough formula to calculate the size of the private region in sectors is (Note that this formula has no scientific or logical reasoning. Use at your own risk.) :

private region size = ((# of disks * 2) + (# of volumes * 5)) * 2

This formula assumes :

. Each physical disk requires 1 disk access record and 1 disk media record
. Each volume averages about 2 plexes and each plex has 1 subdisk
. Each (record + overhead) takes two sectors of private region (This is based on empirical data)

9.2 Startup and Reconfiguration Overhead

By default, a private region is installed on each sliced and simple LSM disks. This has advantages on a small system to protect the configuration database. But on a larger configuration, it becomes redundant and can be a burden on startup time and configuration changes overhead. On startup, each private region is read iteratively in order to reactivate the configuration accurately. The algorithm requires all the private regions to be read and written to add one LSM disk with a private region. So, the number of I/Os needed to start up a disk group with n disks is :

number of I/O = (1 + 2 + ... + n) * 2

And the startup time would be :

startup time = (number of I/O * private region size) / (average disk transfer rate)

Depending on your availability needs, you can reduce the number of LSM disks with private regions to limit the configuration overhead and startup time. You should place the LSM disks with private regions at strategic locations. For example, you might put two copies of private region on each bus or adapter. The rest of the LSM disks can be added into the disk group as nopriv (no private region) disks.

9.3 Implementing a Large Disk Group

As an example, a large disk group with 128 disks and 250 volumes is created here. Note that these commands have not been executed and are not guaranteed to work. They are only provided as a demonstration of the concept.

1. Calculate the private region size needed.

```
private region size = ( ( 128 disks * 2 ) + ( 250 volumes * 5 ) ) * 2
                    = 3012 sectors  ( round up to 3072 sectors )
```

2. Select and create LSM disks with private regions / 1 per SCSI bus.

```
# voldisk -f init rz0c type=simple privlen=3072
# voldisk -f init rz8c type=simple privlen=3072
:
:
# voldisk -f init rz120c type=simple privlen=3072
```

3. Create LSM nopriv disks from the remaining disks.

```
# voldisk -f init rz1c type=nopriv
# voldisk -f init rz2c type=nopriv
# voldisk -f init rz3c type=nopriv
:
:
# voldisk -f init rz7c type=nopriv
# voldisk -f init rz9c type=nopriv
:
:
# voldisk -f init rz127c type=nopriv
```

4. Create disk group and add all disks.

```
# voldg init large_dg rz0c
# disk=1
# while [ $disk -lt 128 ]; do
> voldg -g large_dg adddisk rz${disk}c
> disk=‘expr $disk + 1‘
> done
#
```

5. Create volumes.

```
# volume=1
# while [ $volume -lt 250 ]; do
> volassist -g large_dg make vol vol${volume} 1g
> volume=‘expr $volume + 1‘
> done
#
```

6. Verify the configuration.

```
# volprint -g large_dg -ht
```

9.4 Grouping Disks in Multiple Disk Groups

As an alternative to creating a large disk group, it might be possible to group disks into different disk groups. This reduces the size and overhead of configuration changes and startup time of each disk group. The disadvantage is physical storage from one disk group cannot be used in any other disk group. Therefore, this is only possible if your storage logically sub-divides into smaller volumes. If one large volume is required, a large disk group is the only option.

10 Trouble Shooting

Here are some examples of how to examine LSM on a running system. Trouble shooting LSM is inherently difficult because of the complexity of the component. There are many tools that can aid you in your investigation but they might not be immediately obvious. Some "trial and error" work might be needed.

10.1 Replacing a Failed Device

If a physical disk fails, the LSM disk corresponding to the failing device is marked in the configuration database as failed also. The dm (disk media) entry in the rootdg database is marked as failed. An additional entry is added to show which da (disk access) record was the corresponding physical device.

For example, if physical disk rz0g was defined as dm disk0, if rz0g fails, the da record of rz0g shows "failed" and the dm record disk0 shows "failed : was rz0g". If a device fails, all objects associated with that device are also marked as failed. You have two options to replace the failed device. You can either reconnect the device and bring it back online, or you can replace it with a different device. A command to show all the da/dm records :

```
# voldisk list
```

10.1.1 Reconnecting

You can remove rz0, repair or replace it, and add the device back into the system. In that case, you can add the device rz0 back into it disk group and recover any failed storage if the private region data on the repaired disk is not lost. For instance, if the power supply or a fuse in the drive failed, it usually can be repaired without lost the data. Here are some commands to replace rz0g in disk group dg1 :

```
# voldisk online rz0g
# voldg -g dg1 -k adddisk disk0=rz0g
# volrecover -g dg1 -sb
```

10.1.2 Replacement

If you choose to replace the failed device with a different device, or if the private region data is lost, more steps are required for replacement. The general goal is to provide the disk group with an equivalent amount of physical storage that failed so any objects that rely on the failed device can be reactivated. To do this, the private region has to be initialized before re-integration. Here is an example of replacing rz0g with rz3g :

```
# voldisk -f init rz3g
# voldg -g dg1 -k adddisk disk0=rz3g
# volrecover -g dg1 -sb
```

An example to replace rz0g with a brand new rz0g :

```
# voldisk -f init rz0g
# voldg -g dg1 -k adddisk disk0=rz0g
# volrecover -g dg1 -sb
```

10.2 Failing to Initialize a LSM Disk

The *voldisk init* command is the fundamental command that initializes the private region of a LSM disk. If this command fails, the disk cannot be used in any LSM configurations. Under most circumstances, the command fails because of operational errors. Here are some common cases to check for (you must be super user to run these commands) :

Check for device accessibility. Use the DEC OSF/1 file command, it should report what type of disk it is :

```
# file /dev/rrz0g
/dev/rrz0g: character special (8/6) SCSI #0 RZ26 disk #0 (SCSI ID #0)
```

Check for device accessibility. Read a few block to see if the device is accessible :

```
# dd if=/dev/rz0g of=/dev/null bs=512 count=5
5+0 records in
5+0 records out
```

Check read-only block 0. If *voldisk init* report a write failure, check disklabel for the partition offset. A LSM disk private region cannot include the read-only block 0 :

```
# voldisk -f init rz0a
voldisk: Device rz0a: define failed:
          Disk write failure
```

Check to see if the device is opened by another application or utility. A device that is mounted for other purpose will fail to be initialized. This is a common mistake. Use the DEC OSF/1 df or mount command to see if the disk to be initialized is already mounted. The typical message you get is :

```
# voldisk -f init rz0g
voldisk: Device rz0g: define failed:
          Device path not valid
# df
# mount
```

10.3 Cannot Reuse a Ex-LSM Disk

After a disk is removed from a disk group, if you still cannot use it for other applications, check to see if voldisk has the device open. The complete sequence to remove a rz0g from rootdg is :

```
# voldg -g rootdg rmdisk rz0g
# voldisk rm rz0g
```

10.4 Missing /etc/vol/volboot File

The /etc/vol/volboot file contains the list of disks vold uses to startup after a reboot. If this file is missing or corrupted for any reason, vold cannot restart. A common reason for missing this file is an upgrade or installation of a new baselevel of the operating system. To recover this file, start vold in disable mode, add one disk that you are certain that it is in the configuration, enable vold, and reconstruct the file by adding critical disks into it. If you are not sure of any disks in the original configuration, you can use disklabel to try to find the LSM tags (LSMsimp, LSMpubl, LSMpriv) used to identify LSM disks. Here is an example of how to recover the /etc/vol/volboot file after a reboot. This examples assumes rz3 is a disk in the configuration. After the configuration is reconstructed by vold, the critical disks rz0g and rz7 are added back into /etc/vol/volboot.

```
# voliod set 2
# vold -k -m disable
# voldctl init
# voldctl add disk rz3
# voldctl enable
# volprint -hqt
# voldctl add disk rz0g
# voldctl add disk rz7
```

10.5 Disk Missing on Restart

If some LSM disks in a disk group failed to start with the disk group, you can check for a few common error conditions.

Check accessibility by using the DEC OSF/1 file command. If device is not accessible by the system, do any appropriate action to recover the device and reboot the system. The disk group will recover the missing devices on the next reboot once the devices become accessible. Issuing a file command on a raw partition device should give you the type of the disk. This means the system can access the device.

```
# file /dev/rrz0g
/dev/rrz0g: character special (8/6) SCSI #0 RZ26 disk #0 (SCSI ID #0)
```

Another common problem is a missing disklabel. LSM relies on the disklabel to identify the offset and length of LSM disks. If the disklabel is missing, corrupted, or incorrectly altered, LSM cannot restart the disk. Issue the disklabel command to verify the disklabel is intact and the appropriate LSM tags exist.

```
# disklabel -r rz0
```

10.6 Disabling LSM automatic restart

In the rare event that you need to disable LSM automatic restart, disable LSM startup from /etc/inittab to allow the system to complete the boot process so the problem can be corrected. The procedure is to boot the machine to single user mode, mount the root file system writable, edit the /etc/inittab file to comment out the LSM startup procedure, then continue booting.

```
>>>boot -fl i
:
:
Enter [kernel_name] [option_1 ... option_n]: vmunix
:
:
INIT: SINGLE-USER MODE
# mount -u /
# ed /etc/inittab
:
            comment out the following line
            lsm:23:wait:/sbin/lsmbstartup >/dev/console 2>&1 ##LSM
:
# ^D
```

Remember to uncomment the LSM startup line in /etc/inittab once the problem is corrected. Otherwise, LSM does not start automatically after reboots.

10.7 Process Hang

If you suspect a process hang is related to LSM, it is usually difficult to find the problem but you can try to narrow down the problem by examining the kernel data structure. The general principle of such analysis is to separate the investigation into a few layers and try to eliminate the components one by one. In a typical application, the process issues I/O through the file system to LSM and LSM in turn issues I/O to physical disk drivers on behalf of the application. The five components to investigate are 1) process, 2) file system, 3) LSM, and 4) underlying drivers, 5) hardware. The following example assumes that something is wrong with the hardware to show the different steps in the analysis.

10.7.1 Process State

Examine the process state by using the ps command and dbx. If a process is waiting for a synchronous I/O to complete, the ps command will show the process state as U (Uninterruptable). Although process state U does not necessarily imply a sync. I/O wait, it is often the case. Issue the following command and examine the state of the process in question.

```
# ps aux
```

If the process stays in an uninterruptable state and does not incur any cpu time, you need to use dbx to get more information about the process. Note the process id from the ps command. Invoke dbx and examine the stack of the process. If the final call frame on the top of the stack is a biowait, then it is probably waiting for an I/O to complete. Find out which device I/O is outstanding on. You can go through the vnode pointer or buf struct if they are available. On a optimized kernel, this might not be possible. If you cannot trace the vnode pointer or buf struct, then guess the device.

```
# dbx -k /vmunix /dev/mem
:
(dbx) set $pid=123
(dbx) trace
:
:
```

10.7.2 File System State

While in dbx, examine the file system information if available. If the top frame on the process stack is a file system lock_write or other blocking call frame, it is possible that the file system is hung due to other reasons.

10.7.3 LSM layer

If you have an LSM volume you are interested in examining, you can use dbx to look at the kernel structure. First determine the device minor number of the LSM volume. For instance, in the following example, vol01 has a minor number of 5.

```
# file /dev/vol/*
/dev/vol/rootdg:        directory
/dev/vol/vol01: block special (40/5)
/dev/vol/vol02: block special (40/6)
/dev/vol/vol03: block special (40/7)
#
```

Once you have the minor number, invoke dbx on the running kernel and print the volume structure indexed by the minor number. For example, for vol01 with minor number 5 :

```
# dbx -k /vmunix /dev/mem
:
(dbx) p voltab[5]
struct {
            vt_vp = 0xffffffff895d9c00
            vt_flags = 0
            vt_active = 1
            vt_operation = 0
            vt_logcb = (nil)
:
(dbx)
```

The vt_active field usually indicates how many I/O are outstanding on the volume. If the count is non-zero, I/Os are stalled either in the LSM layer or in the underlying driver. Try to determine the state of the volume and its associated LSM disks. The next step is to examine the physical device driver.

In the case where the application is using asynchronous I/O, you must use the process dependent method to determine whether the process is waiting on an incomplete I/O indefinitely.

10.7.4 Underlying Drivers

LSM issues physical I/O on behalf of the application I/O. Depending on the volume layout, I/O may go to one or more physical devices per application I/O. Examine the kernel structure of the drivers. For example, if vol01 encompasses SCSI devices rz8 and rz9, you can use the following commands to look at the driver structures for rz8 and rz9. (Use the drive_number * 8 as index.) This example only works with SCSI devices.

```
# dbx -k /vmunix /dev/mem
:
(dbx) p *pdrv_unit_table[64].pu_device
struct {
            :
            pd_active_ccb = 0
            pd_que_depth = 0
            :
}
(dbx) p *pdrv_unit_table[72].pu_device
struct {
            :
            pd_active_ccb = 1
            pd_que_depth = 0
            :
}
(dbx)
```

The pd_active_ccb field indicates how many I/Os the driver is processing on this unit. In this example, there are no I/Os outstanding on rz8 and one I/O outstanding on rz9.

10.7.5 Hardware

Look at the hardware associated with the volume to see if anything is obviously wrong. Obtain the physical drives associated with the volume and examine each device. For instance, if you see a drive access light stays on and there is no apparent read/write activity, there is something wrong with the state of the controller or the drive. Check the power supplies and cables of the problem devices. Consult experts in these areas to resolve any problems.

11 LSM & UFS

From the UFS stand point, a LSM volume is similar to a physical disk partition. Volume attributes like mirroring and striping have no configuration significance to UFS. I/O going to a file in a file system that resides on a LSM volume goes through the regular code paths, VFS, UFS, UBC, until it gets to the device. At the device level, the buf struct is passed to the LSM driver instead of the SCSI/CAM or MSCP driver.

11.1 Making a UFS file system on a LSM volume

There are a few minor differences between a physical device partition and a LSM volume. 1) A LSM volume does not have a disklabel. 2) Block 0 of a LSM volume is not write protected. 3) A LSM volume is constructed with a variety of physical devices and the physical devices can migrate in and out of a LSM volume without affecting the application view of the volume. As a result, the physical geometry is indeterminate. For this reason, you have to supply the geometry when you make a UFS file system on a LSM volume. In the following example, a 5 GB volume is created by combining a DSA drive ra5, and a SCSI drive rz3. rz74 is used to supply the default track size and other geometry information.

```
# volassist make vol01 5g ra5 rz3g
# newfs /dev/rvol/rootdg/vol01 rz74
# mount /dev/vol/rootdg/vol01 /ufs1
```

Here is another example of making a UFS file system on a mirrored LSM volume using three rz26s. Note that the length specified in newfs is optional.

```
# volassist make vol02 1g mirror=yes rz0 rz1 rz2&
# volprint -v -F "%len" vol02
2097152
# newfs -s 2097152 /dev/rvol/vol02 rz26
:
# mount /dev/vol/vol02 /ufs2
```

Here is an examples of making a UFS file system on a striped LSM volume of 6 rz28s.

```
# volassist make vol03 12g layout=stripe nstripe=6
# newfs /dev/rvol/vol03 rz28
:
# mount /dev/vol/vol03 /ufs3
```

11.2 Checking and Repairing a UFS file system on a LSM volume

Use the fsck command as usual on the raw volume device. For example :

```
# fsck -p /dev/rvol/vol03
```

11.3 "Honey I Shrunk the Volumes"

LSM volumes can be expanded and shrunk while the volume is enabled or disabled. If a UFS file system resides on a LSM volume, expansion and shrinking of the LSM volume should be avoided. Since UFS does not take advantage of a device expansion after newfs completes, volume expansion has no effect of the UFS file system on the volume. If the volume is shrunk, the file system will be corrupted since part of the file system and its data will be missing.

In the case where the volume is accidentally shrunk, you can attempt to recover the data. Disable access to the volume as soon as possible. Unmount the file system. Locate the LSM disk the lost data was on. Expand the volume using that particular LSM disk (and offset if appropriate). This is not a safe method to recover the data. The only correct way to recover is to restore from tape.

In the following example, the volume consists of 300,000 blocks from three LSM disks, rz0, rz1, rz2. The volume was accidentally shrunk by 2,000 blocks.

```
# volassist make vol01 300000 rz0 rz1 rz2
# newfs /dev/rvol/vol01 rz26
# mount /dev/vol/vol01 /ufs1
:
: time goes by
:
# volassist shrinkby vol01 2000
# stop_application
# umount /ufs1
# volassist growby vol01 2000 rz2
# fsck -op /dev/rvol/vol01
```

11.4 Pseudo Online Backup

LSM provides a snapshot mechanism to reduce the backup overhead. See the LSM system administration guide for operational details. The general idea is to make a mirror copy of the volume using the snapshot mechanism. Once the mirror copy completes, the copy is kept in sync with the volume by acting as a mirror plex. When the system administration is ready to perform the backup, the file system can be dismounted, the snapshot plex is detached to make a separate copy of the volume for backup. The original volume can be brought back on-line as soon as the snapshot copy is separated. So, while the backup is running on the snapshot copy, the user data is available to applications.

This is not a true on-line backup but it reduces the off-line time dramatically. Please refer to the volassist man page and the LSM System Administration Guide for more details.

12 LSM & AdvFS

12.1 Comparing LSM & AdvFS

The current AdvFS release has certain key features. Among them, the most obvious ones are : 1) Provides log-based recovery. 2) Provides container approach to storage. 3) Provides striping at file set level. 4) On-line backup.

Log-based recovery is an important feature to modern file systems. With this approach, file system meta-data is logged in non-volatile memory before it is modified in the actual storage container. In the case a system crash happens while the storage container is being modified, the meta-data consistency can be recovered quickly using the non-volatile log. This is a key feature of DEC OSF/1.

AdvFS uses a domain as the storage container. Physical storage is added to the domain as AdvFS volumes. This storage space is managed by AdvFS to create mountable file systems called filesets. Each fileset can grow until the domain is full. AdvFS volumes can be added and removed while filesets are mounted and actively used by applications. When an AdvFS volume is removed from a domain, all the storage is automatically moved to the remaining AdvFS volumes.

Fileset striping works well but requires additional setup which is complex.

AdvFS has a powerful clone-and-backup facility implemented with vdump and vrestore. This is a true on-line backup method. The integrity of the backup is guaranteed even if it is done while concurrent user access is going to the fileset. And it is much more efficient than LSM backup since it only backs up the populated area of the domain. LSM has to backup all blocks on all AdvFS volumes since it does not understand what areas in the AdvFS domain are in-use.

The current LSM release has certain key features. 1) Provides container approach to storage. 2) Precious storage manipulation. 3) Provides mirroring. 3) Provides easy striping.

LSM also uses a storage container approach called disk groups. Physical storage is added to the disk group as LSM disks. These LSM disks can be used to create LSM volumes. LSM volumes appear as physical partitions to the operating system. A set of sensible defaults are used to create LSM volumes if you choose not to specify any details. On the other hand, great control over the storage allocation is possible through a set of comprehensive parameters used when creating or changing LSM volumes.

Concatenated, mirrored, striped, and mirrored/striped volumes can be created and manipulated. Performance analysis and trace information is also available.

12.2 Recommendation on AdvFS vs. LSM

In my LSM biased personal opinion, you should use AdvFS for its log-based recovery feature, the capability to allow filesets to grow and shrink freely within a AdvFS domain, and its powerful on-line backup feature. You should use LSM for storage management of the AdvFS volumes that make up a domain. The combination of the products gives you a flexible and powerful storage subsystem.

12.3 Using LSM Volumes in a Domain

LSM volumes can be used in an AdvFS domain as any disk partition. You first create the LSM volumes with the desired attributes and then use AdvFS commands to add them into an AdvFS domain as AdvFS volumes. When a device is added to the domain, a soft link is created in the AdvFS domain directory tree with the volume name that points to the pull access path of the LSM volume.

Here is an example of how to create a 1 GB AdvFS domain with 2 LSM volumes :

```
# volassist make vol01 500m
# volassist make vol02 500m
# mkfdmn /dev/vol/vol01 domain
# addvol /dev/vol/vol02 domain
# mkfset domain fset1
```

12.4 Restriction on LSM volumes as AdvFS volumes

The only known restriction at this time is LSM volume names must be unique within a AdvFS domain. In LSM, volume names can be the same provided the volumes are in different disk groups and therefore has different full access paths. For instance, vol01 in dg1 has access path /dev/vol/dg1/vol01 and vol01 in dg2 has access path /dev/vol/dg2/vol01.

AdvFS domain tree uses only the last token, the volume name along in the domain tree. Therefore, two LSM volumes with the same name would appear to be the same to AdvFS and cannot be used in the same AdvFS domain.

12.5 Managing "Property" Domains

Currently, AdvFS does not recognize different properties of its domain volumes. The only way to assure a given fileset has a certain storage attribute is to base it on an AdvFS domain that contains LSM volumes having matching properties. For instance, in the following example, four different kinds of domains are created: A plain domain, a mirrored domain, a striped domain, and a mirrored-striped domain. Filesets requiring different attributes can be created on the matching domain.

Create a 1 GB AdvFS domain with 2 LSM volumes :

```
# volassist make vol01 500m
# volassist make vol02 500m
# mkfdmn /dev/vol/vol01 cheap_domain
# addvol /dev/vol/vol02 cheap_domain
# mkfset cheap_domain cheap_fset1
```

Create a 1 GB AdvFS domain with mirrored LSM volumes :

```
#volassist make vol03 500m mirror=yes
# volassist make vol04 500m mirror=yes nmirror=3
# mkfdmn /dev/vol/vol03 safe_domain
# addvol /dev/vol/vol04 safe_domain
# mkfset safe_domain safe_fset1
```

Create a 5 GB AdvFS domain with striped LSM volumes :

```
#volassist make vol05 2g layout=stripe nstripe=6
# volassist make vol06 3g layout=stripe nstripe=3
# mkfdmn /dev/vol/vol05 fast_domain
# addvol /dev/vol/vol06 fast_domain
# mkfset fast_domain fast_fset1
```

Create a 5 GB AdvFS domain with mirrored-striped volumes :

```
# volassist make vol07 1g layout=stripe nstripe=4
# volassist mirror vol07 layout=stripe nstripe=4 &
# volassist make vol08 1g layout=stripe nstripe=4
# volassist mirror vol08 layout=stripe nstripe=4 &
# mkfdmn /dev/vol/vol07 fast_and_safe_but_expensive_domain
# addvol /dev/vol/vol08 fast_and_safe_but_expensive_domain
# mkfset fast_and_safe_but_expensive_domain fast_and_safe_but_expensive_fset1
```

Use filesets according to application needs :

```
# mount -t advfs fast_and_safe_but_expensive_fset1 /payroll
# mount -t advfs fast_fset1 /build_object_area
# mount -t advfs safe_fset1 /source_tree
# mount -t advfs cheap_fset1 /games
```

Since all the storage in a mirrored domain is mirrored regardless of whether the storage is in use by a fileset or not, an alternative is to create a smaller domain initially and add volumes when needed. For instance, a striped domain can be created with just 2 GB instead of 5 GB. Later on, if it runs out of room, add a second striped LSM volume into the striped domain. Once the additional storage is available, the filesets can grow within the domain. Be careful when adding additional volumes into property domains. The property of the LSM volume must match the property of the domain.

If it is discovered that any LSM volumes in a domain have mismatched properties, replace the LSM volume with matching property by using AdvFS commands or convert the LSM volume to the appropriate property volume using LSM commands. For example, if a mirrored domain has a non-mirrored volume, you can use either option to correct the situation.

Mirrored domain created with incorrect volumes :

```
# volassist make vol01 1g mirror=yes
# volassist make vol02 1g
# mkfdmn /dev/vol/vol01 mirror_domain
# addvol /dev/vol/vol02 mirror_domain
:
```

Correcting with AdvFS commands :

```
# volassist make vol03 1g mirror=yes
# addvol /dev/vol/vol03 mirror_domain
# rmvol /dev/vol/vol02 mirror_domain
```

Correcting with LSM commands :

```
# volassist mirror vol02
```

13 LSM & RAID5

13.1 RAID Terminology

There are a few translations of RAID. The original one I believe is Redundant Arrays of Inexpensive Disks. They are also called Redundant Arrays of Independent Disks, etc. The basic idea is to add a parity disk to an array of striped disks so that if any one device fails, the data can be reconstructed from the parity. By doing this, the array is protected again any single disk failure. This is what is called RAID5.

In addition to RAID5, other RAID levels are defined. The most common ones are RAID0 and RAID1. RAID0 is striping. RAID1 is mirroring or shadowing. These terms are use interchangeably in this chapter.

13.2 Comparing LSM and Hardware RAID5

If you need RAID5, buy the hardware boxes. LSM does not support RAID5 currently. In addition to RAID5, most of the RAID5 boxes provides striping and mirroring. Here is a discussion on the difference between a hardware box and using LSM as a software solution.

From an availability stand point, RAID5 hardware boxes can have multiple adapters and power supplies to increase its availability. And it is supported by ASE in failover environments. From a performance stand point, the hardware handles all the copies, recoveries associated with RAID5, mirroring, and striping, and hence reduces the system overhead. From a storage management stand point, you can mix and match the drives in the box in units of one drive to create RAID5, mirror, or stripe sets. Some controllers can divide drives and use different sections for different logical units.

LSM allows you the flexibility to mix and match disks and sections of disks from different architecture, across controllers and buses. From an availability stand point, LSM gives you a high degree of configuration flexibility. You can allocate storage on different controllers or buses to increase availability. LSM is also supported in the ASE environment. From a performance stand point, again, LSM gives you the flexibility of multiple controllers and multiple buses. But the system has to incur the overhead of managing the configuration. In addition, LSM allows you to eliminate bus bottle necks. RAID5 boxes are usually attached to one bus and hence limited to the bus bandwidth. From a storage management stand point, LSM allows you more options from data manipulation to online-backups. For instance, you can move data from one disk on one controller to another disk on another controller while the data is online. From hardware preservation stand point, if you have existing hardware and want to utilize them, LSM is a viable solution.

Lastly and most importantly, LSM and hardware RAID5 can be used together. This combination can give high I/O bandwidth while off-loading storage overhead from the CPU to the RAID5 hardware.

13.3 Recommendation on Hardware RAID5 vs. LSM

13.3.1 Need RAID5

If you need RAID5, buy a RAID5 box. But you might still want to put the virtual unit under LSM management to take advantage of some storage management features.

13.3.2 Existing Hardware

If you have to work with existing disk hardware and don't want to replace them with new RAID5 boxes, use LSM.

13.3.3 New Setup

If you are setting up a new platform, you may use either one. See the Comparison Section.

13.3.4 Multiple Buses

If you need special availability or performance characteristics, you may need LSM in order to utilize multiple buses and controllers since a RAID5 box is limited to one bus.

13.4 Using RAID5 Volumes in LSM Disk Group

Use virtual units and physical disks. Add them into the LSM disk group as LSM disks. You can use any regular LSM commands once you use the RAID5 utilities to form a virtual unit.

To set up a disk group and a volume with RAID5 virtual unit :

```
# box_specific_create_command rza40
# disklabel -e rza40
:
: Edit offset of partition c from 0 to 1 and shrink size by 1.
:
# voldisk -f init rza40
# voldg init raid_dg rza40
# volassist -g raid_dg make vol01 5g rza40
```

To setup a striped volume based on RAID5 virtual units :

```
# box_specific_create_command rza40
# box_specific_create_command rzb40
# box_specific_create_command rzc40
# box_specific_create_command rzd40
# box_specific_create_command rze40
# box_specific_create_command rzf40
# voldiskadd rza40 (add as raid5_1)
# voldiskadd rzb40 (add as raid5_2)
# voldiskadd rzc40 (add as raid5_3)
# voldiskadd rzd40 (add as raid5_4)
# voldiskadd rze40 (add as raid5_5)
# voldiskadd rzf40 (add as raid5_6)
# volassist make vol01 24g layout=stripe nstripe=6 raid5_1 raid5_2 raid5_3 \
> raid5_4 raid5_5 raid5_6
#
```

14 LSM & ASE

14.1 Using LSM in ASE

LSM is supported by the ASE product. In general LSM configuration setup and changes are done outside of ASE. To maintain failover integrity, ASE knows about the LSM volume dependencies and configurations once it is added into an ASE service. The ASE Manager interactively queries you about the LSM configurations during the setup phase of an ASE service.

14.2 LSM configuration Changes

If changes are made to a LSM configuration in a ASE service, you must run the ASE manager, asemgr to update the ASE configuration database. Please refer to the ASE documentation for details.

14.3 Special ASE LSM requirement

Each ASE member that is configured to serve any service with LSM as underlying storage must have LSM installed and automatically started after a reboot. A LSM rootdg must exists on each member independent of any shared disk groups. This is necessary to ensure LSM can start on a ASE member even if no ASE/LSM service is active on that member.

14.4 Failing LSM Disks in ASE

In a very simplified version, if a device becomes inaccessible and ASE detects that the device is available on another ASE member, the service associated with the inaccessible device is failed over to that ASE member. With LSM mirrored volumes, the logic changes slightly. If a device becomes inaccessible on one ASE member, the ASE director queries each LSM volume to see if it has become disabled by the inaccessible device. Only if at least one LSM volume becomes disabled, ASE attempts to fail over the service. In other words, if one device in a mirrored volume becomes inaccessible but the LSM volume remains available, no fail over is attempted by ASE.

14.5 LSM Disk Group in ASE Failover

An LSM disk group is a group of physical disk partitions with a self-describing configuration database. This is the unit an ASE service can failover between ASE members. Configuration changes are done by the ASE member on which the disk group is active. Since the configuration database is completely contained in the disk group, the changes are reflected on the ASE member when the service is brought on-line.

In a very simplified version, ASE reserves all physical devices associated with a service before starting or restarting an ASE service. In order for the LSM disk group to move among ASE members independently, all physical devices in a disk group must be unique to that disk group. In addition, a disk group must not span more than one ASE service in order for the services to move among ASE members freely. Since disks in the rootdg disk group is specific to each ASE member, they cannot failover among ASE members.

14.6 Triple Failure Partitioning

Consider a rare triple failure illustrated by the diagram below. In this scenario, ASE Member A has the service online. The first SCSI cable has a failure at Point 1. This caused LSM to change the status of Plex 1 to IOFAIL and continue to update Volume 1. The LSM configuration database is only updated on Plex2 since Plex1 is not accessible. The second SCSI cable has a failure at Point 2. This does not affect Member A for the moment. Now consider a system crash of Member A and the service is failed over to Member B. Since Member B cannot access Plex2, Plex1 configuration database is used. Volume 1 becomes online on Member B with Plex2. The updates to Volume1 between the first SCSI cable failure and Member A system crash is lost. The triple failure is SCSI cable 1 failure, SCSI cable 2 failure, and system crash of Member A. This is an extremely rare event but nonetheless possible.

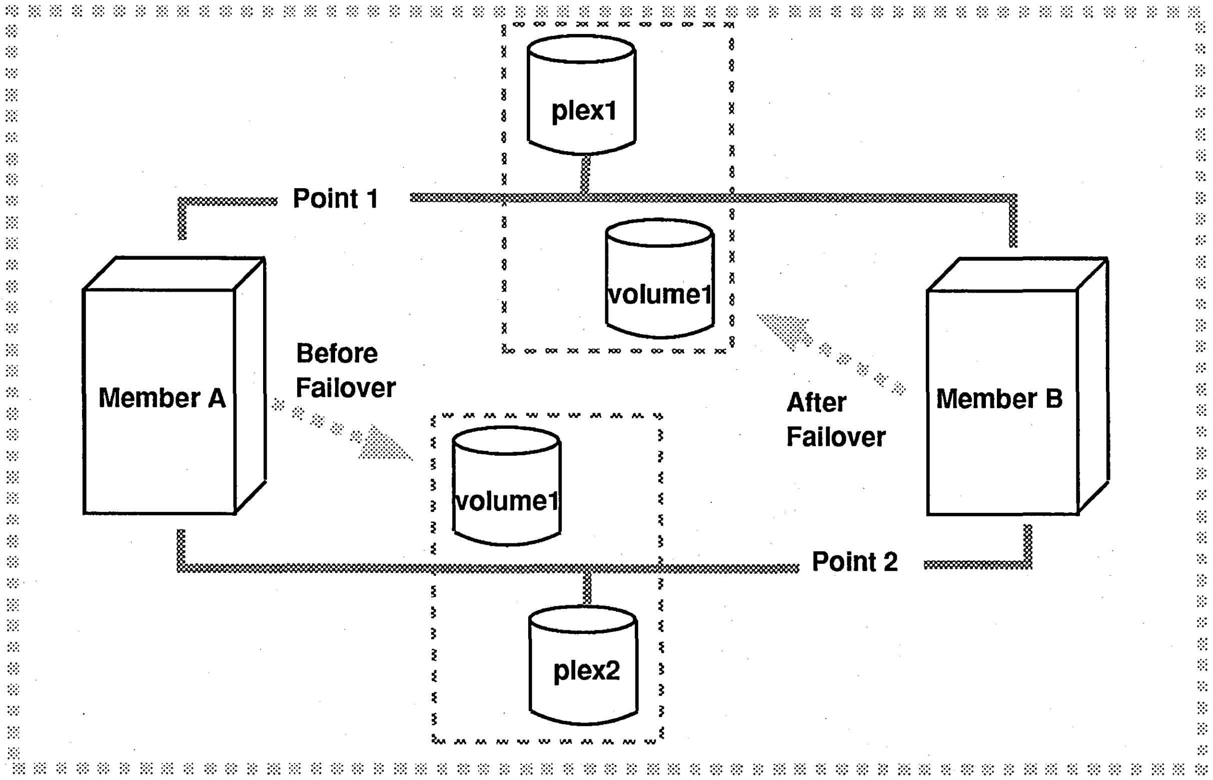


Figure 5 Triple Failure Partitioning

The scenario should not be advertised. But if you feel your customers application is sensitive enough to worry about this case, there is an ASE switch that you can use to disable the failover under this circumstance. This switch is ASE_PARTIAL_MIRRORING. If set to "off", failover does not occur is one of the plexes is not available. Unfortunately, the side effect of this switch set to "off" is if a plex has failed, failover will not occur regardless of whether it is the triple failure partitioning scenario or not.

You can increase availability if you set ASE_PARTIAL_MIRRORING to "off" on all but one member. This way, as long as this member is available, the service with a partial mirror can run. The triple failure partitioning cannot happen with this setup because only one member can run the service with a partial mirror. If the member with ASE_PARTIAL_MIRRORING set to "on" is not available, then the service becomes unavailable if a service with a partial mirror needs to be relocated for any reason.