The Cuddletech Veritas Volume Manager Series

Advanced Veritas Theory

Ben Rockwood
CDLTK
Cuddletech

benr@cuddletech.com

VxVM 3.x adds the powerful capability to create "layered volumes", the ability to create a volume inside a volume. Sadly, practical coverage of creating such volumes has been sparse. In this tutorial we discuse the topic at length and walk thru several examples of creating layered volumes from start to finish.

Welcome to the Real World

Up until this point you have been learning how to put Veritas objects together. It is hoped that by now you have a good solid grasp of how these objects are put together to form volumes. Now, we're going to quicken the pace and learn some of the things you be dealing with in every day volume management. I'm not going to cover any of these concepts in their entirety, but rather going to give you a pedestal to jump off of. Remember that there is no way for anyone to teach you all that you want to know about any particular topic, especially Veritas. You may then wonder why this course is entitled "advanced", this is because the following topics are a gateway into your own study into the Veritas volume manager. From these introductions you will go on to man pages and product guides. Absorb the following and you should be well set to move on into a excellent career as an Enterprise Storage Administrator.

Mirroring

As pointed out earlier mirroring is a trivial task in Veritas. Unlike other volume managers we can mirror a volume of any size in only one line. A plex is a mirror, remember saying that? Just add a plex.

A little example. We've got a 50G striped volume containing only one plex (duh, it's unmirrored!). Our boss tells us she wants to get that 50G volume mirrored for safety. We tell her not to worry, you'll be done before she's back from lunch. You'd then add some disks (if they aren't there) to the system and initialize them into Veritas. Once you'd created your VM disks we'd want to build a striped plex, using

vxmake or vxmirror, which ever you prefer. There is a special point to be made here: when adding a second plex to a volume, thereby creating a mirror, all of the plexes the same volume should be the same layout. Since we have a striped volume, we therefore must have a striped plex (because the plex handles the layout, not the volume) which means we need to create a striped plex from our VM disks. Once our striped plex exists we only need to associate it with the volume we want to mirror. The instant that you associate the plex you'll notice that vxprint reports the volume and plex as "SYNC", which means that the data from the first plex is pouring onto our new mirror. Once the sync finishes your fully mirrored! That easy. And if you can get the disks on the system without a reboot (like on Solaris) you could have done all this without a second of downtime!

Mirror Juggling

Right up from I want to say that I DO NOT recommend this practice for the squeamish. If your like me and like loosing sleep at night, and like scaring the pants off Sun and Veritas tech support, then go for it.

Mirror Juggling is the term I dreamed up for one of my favorite pastimes. The idea is to rebuild existing volumes by attaching a plex (mirror) to the volume, sync'ing the plexes, and then detaching the original plex from the volume. At that point I'd destroy the original plex, and rebuild it from scratch to my design. Once the plex was rebuilt the way I wanted it, I'd attach the plex back to the volume, and let it sync. After the sync is done I can remove my temporary plex. See the idea here? The mental image I get when doing this is of water (representing the data) being rolled around in a box (representing the volume). As I roll the volume I slosh the data along with it, rebuilding behind it.

This process is pretty simple in concept, but horribly scary... on a production system this isn't for people with heart problems. But, if you hadn't guessed, the reason I'd do something as risky as this is because I can completely rebuild volumes from scratch on a hot system. NO DOWNTIME! Generally, in the environments I've worked, the systems running VxVM are the production systems that are expected to deliver 99.999% annual uptime, which doesn't lend for fixing old problems, and design flaws. Mirror Juggling is the solution I came up with.

The key that we're exploiting through this process is Veritas's ability to sync and contain plexes that aren't identical. I once used this method to fix a plex that contained 10 subdisks, yet only was striped to only use 5 columns. In this situation I had 5 disks that we're being utilized. Rather than use a cheap hack I want to rebuild, so I took on of my mirrors offline, removed the plex, then the subdisks until I had the plex broken down into it's VM Disks. From there I created subdisks, and then specified the striping model I wanted (10 columns with a 256k stripe width) to create the plex. Once my plex was the way I wanted it I reattached it to the volume and let it sync. Once it was sync'ed I took the other mirror offline and did the same to it. Afterwords I had only minimal clean up to do (which in this case involved expanding the filesystem to use all the extra space I had!). The point was that this was a production system, if it went down people went home, and I wasn't scheduled to get a downtime for 6 weeks. I did everything during a slow time with 0 downtime.

Spares, Hot Relocation, and Disk Reservation

You've probably seen by now that there are two different worlds in Veritas depending on whether you use VxAssist or VxMake. As you've seen, when you use VxAssist disks are just "magically" allocated

the way you ask VxAssist to do so. Where do these disks come from? The spare pool. When initializing a disk you assign the disk to a disk group, and at that point it's made a member of the spare pool for the disk group. For the VxMake fan's out there, we'd simply say that it's not being used yet. When VxAssist preforms an operation that requires using a new disk it pulls a disk from the spare pool, creates a subdisk, and does what ever it needs to do. In some situations you might not want specific disks to just get freely yanked into service, maybe it had problems at some point, or you are saving it for something else. Whatever the reason, if you don't want to mark a disk "off limits" you can simply reserve it. This is done with vxedit. When you reserve a disk, your actually just marking a flag on the VM Disk. The syntax is:

```
vxedit set reserve=on <diskname>
```

To un-reserve it, the syntax would be:

```
vxedit set reserve=no <diskname>
```

Note that reserves can will be ignored if you assign it by name, as in the following example:

```
vxassist make vol01 500m disk03
```

Even if disk03 was reserved, it will be used.

Hot Relocation is also know as Hot Sparing. When a VM Disk is make a hot spare, it become a life raft for the disk group it is assigned to. If VxVM detects, or suspects a failure (or if a drive flat dies) the drive will be logically replaced with the hot spare.

And example, if you had a mirrored volume and one of the disks failed, normally you'd loose on of the mirrors (the plex would fail) and you'd have lost your redundancy, so if another disk failed on the other mirror (assuming you only had 2 mirrors) the volume would be lost. However, if you had a hot spare assigned to the disk group, when the disk failed, the hot spare would immediately take it's place and the mirror would re-sync. What does this do? Buy you time. This simply gives you more time to respond to the problem and replace the failed component. Note that there is no limit to the number of spares you can assigned to each disk group, but that hot spares will only relocate failed disks in it's diskgroup. Sparing is enabled by either answering "yes" the question "Is this disk a hot spare?" during disk initialization, or after it's initialized by using the syntax:

```
vxassist set spare=on <diskname>
```

Hot sparing can be disabled for a disk, by using:

```
vxassist set spare=off <diskname>
```

A handy side note, hot sparing also reserves the disk so it can not be used as a member of the spare disk pool. Often when I want to make disks "hands off", I'll make them hot spares instead of reserving them, at least then their being useful.

Dirty Region Logging (aka: DRL)

Dirty Region Logging is mentioned often in the VxVM manuals but poorly explained. For most people I've talked to they had a difficult time trying to get a grasp of what it is. The way I like to introduce DRL

to new users is to ask them to think of DRL as journaling (like a journaling filesystem) for volumes. Now, in fact this isn't exactly correct, but it's a good starting point.

DRL is a fault recovery mechanism. When logging is enabled for a volume a bitmap is created representing the volume it's attached to. When a volume region is modified it's corresponding bit in the bitmap is marked dirty BEFORE the write is made to the volume. A region is a logical portion of the volume that is defined and used by DRL, which is considered more efficient than block-by-block logging. By default, a region is defined as 1024 sectors. After the write operation completes to the volume DRL doesn't remove the dirty bit. The bit isn't removed until it is the "least recently used". By default, the maximum number of dirty regions a system can have at one time is 2048. Therefore once 2048 regions are marked dirty the first region marked dirty will be cleared. If a region is to be marked dirty, but a dirty bit is already set the previous dirty bit is let stand, and the write is made. Note that DRL is only effective for mirrored volumes.

The big idea here is that should your system crash for one reason or another the system would normally have to sync all the mirrors in the volume before making the volume available, all because the system doesn't know what changed, so it has to check everything. However DRL provides a mechanism by which to know exactly what has changed, therefore speeding up the sync'ing process by only sync'ing the "dirty" regions. Handy huh?

Okey, here's the big remaining question most people have: "If data is written to all mirrors in parallel, and the system crashes in the middle of a write there the data in the region is still going to be mangled. How is DRL going to make sure the valid data is in place?" And the answer is that DRL isn't intended to protect the validity of your data, but rather that all the data is consistent. Let me explain a bit more.

Remember that a volume manager doesn't understand the data it contains, it's a bit bucket. The volume manager's whole purpose in life it to protect what you write to it, and make sure that it's available to you, NOT to verify that your data is what you intended it to be. Often this is forgotten. So, back to DRL, if the system crashes in the middle of a write, all the region the data was being written to may not be identical. In theory this wouldn't be a problem, but I/O slow downs, disk speeds, and other performance factors make parallel data writes happen at slightly different times. Therefore, each mirror may have different data in region that was being written. DRL simply makes sure that all the data in the dirty region is identical, regardless is the data is any good or not. If the regions where not identical across mirrors two reads to the same region of the volume could have different results. DRL at least makes sure that all the regions have the same data, by sync'ing the regions from the preferred plex. The data might be crap, but at least it's consistent crap!

To enable DRL for a volume you can simply use vxassist:

```
vxassist addlog <volumename>
```

VxAssist will create a subdisk, mark it as a log disk, and then attach it to a plex. And that's it! A really simple process.

Note that log subdisks length corresponds to the length of the logged volume. The log subdisk should contain 1 sector for every 1G of volume. So if your log subdisk is 80 sectors long, and you decide to grow the volume to 120G your going to need to grow the DRL as well.

Also, logs can be mirrored simply by adding another log subdisk to the volume. But also know that you can only have 1 log subdisk per plex.

DRL can not be used on root volumes, including: rootvol, standvol, swapvol, and usr.

Dynamic Multi Pathing (aka: DMP)

Dynamic Multi Pathing is used to increase disk availability. This is available when using disk arrays that allow for connection to multiple controllers. DMP does two big things for us. Firstly, it gives us multiple "paths" or physical connections to each disk in an array, so if one path fails (like if a controller, cable, GBIC, I/O board, etc. dies) the disk is still accessible. Secondly, because there are multiple I/O paths to DMP disks it allows for load balancing by splitting I/O across each path, in turn lowering load on system controllers, busses, etc. DMP can be manipulated by using the tool "vxdmpadm". This tool allows DMP to be enabled, disabled, and for various DMP operations such as disabling paths for maintenance. To see what paths are available to a given disk use the vxdisk command. (syntax: vxdisk list <devname>) DMP nice in the fact that it rarely requires administrator intervention. Quite simply, if your system is setup correctly it "just works". For further information, see the Veritas Getting Started Guide, and the vxdiskadm(1m) man page.

Unencapsulating Your Root Disk

There are two methods for this. The preferable choice is to use the command "vxunroot". If you can't do this for some reason, or you need to repair a system that has crashed, you can manually un-encapsulate the disk using the following steps:

```
1) Boot off the cdrom
     # boot cdrom -sw

2) Mount the root file system.  Your root slice could be different.
     # mount /dev/dsk/c0t0d0s0 /a

3) Edit /a/etc/system
     Using "*" comment out the lines between VXVM BEGIN and VXVM END
     NOTE: There might be a file /a/etc/system.prevm
           That could be used.  The lines look like the following:
            *rootdev:/pseudo/vxio@0:0
            *set vxio:vol_rootdev_is_volume=1

4) Edit /a/etc/vfstab
     Need to comment out the line that references the Volume Manager volumes.
     NOTE: There might be a file /a/etc/vfstab.prevm

4a) Using the information in the vfstab file change the root device to system
      root disk.  ie.

      /dev/dsk/c0t0d0s0   /dev/rdsk/c0t0d0s0  /  ufs  1  no   -

      There are lines within the vfstab starting with "NOTE:" that
      indicates the original boot device.  Use this information to recreate the
      original vfstab file.

      There are line similar to this:

     #NOTE: volume rootvol (/) encapsulated partition c0t0d0s0
```

```
    4b) Do the same thing for the other encapsulated volumes (ie. /usr , /opt,
  ..etc)

5) Unmount /a, and reboot.
```

A Brief Discussion of VxFS

Most of us, at this point are use to building fileystems with UFS. However Veritas offers the Veritas File System. A journaling filesystem with performance advantage over UFS. My favorite use of VxFS, however, is that very large filesystems can be created very quickly. This is because... well, I'll finish that later, but you don't have to wait minutes or hours sometimes to let the inode tables build. There are some special ways of working with VxFS, however. The two situations are in building filesystems, and fsck'ing filesystems. This is how you should do it:

```
a) Building a VxFS Filesystem:
 /usr/lib/fs/vxfs/mkfs -F vxfs -o largefiles /dev/vx/rdsk/exampledg/vol01
 -Sample Output:
    version 4 layout
    35363560 sectors, 4420445 blocks of size 4096, log size 256 blocks
    unlimited inodes, largefiles supported
    4420445 data blocks, 4419853 free data blocks
    135 allocation units of 32768 blocks, 32768 data blocks
    last allocation unit has 29533 data blocks


b) FSCK'ing a VxFS Filesystem:
 fsck -F vxfs -o full,nolog /dev/vx/rdsk/exampledg/vol01
 -Sample Output:
 pass0 - checking structural files
 pass1 - checking inode sanity and blocks
 pass2 - checking directory linkage
 pass3 - checking reference counts
 pass4 - checking resource maps
 OK to clear log? (ynq)y
 set state to CLEAN? (ynq)y
 -Note: You can _try_ using fsck with just the -F option, and ditching
 the full,nolog options, but if you think the FS might really
 be messed up you need 'em.  Without them fsck will do something
 like this:
 # fsck -F vxfs /dev/vx/rdsk/prod6gr/saveusr
 file system is clean - log replay is not required
```

Auto/Online Relayout

```
To be written:
 -Volume Status, Startablity and Error Handling
 -Snapshots and Online Backups
```