USENIX Association

Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference

Boston, Massachusetts, USA
June 25–30, 2001

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Volume Managers in Linux

David Teigland
Heinz Mauelshagen
Sistina Software, Inc.
http://www.sistina.com

Abstract

A volume manager is a subsystem for online disk storage management which has become a de-facto standard across UNIX implementations and is a serious enabler for Linux in the enterprise computing area. It adds an additional layer between the physical peripherals and the I/O interface in the kernel to present a logical view of disks, unlike current partition schemes where disks are divided into fixed-size sections.

In addition to providing a logical level of management, a volume manager will often implement one or more levels of software RAID to improve performance or reliability. Advanced logical management tools and software RAID are the specialties of the Logical Volume Manager (LVM) and Multiple Devices (MD) drivers respectively. These are the two most widely used Linux volume managers today.

This paper describes the current technologies available in Linux and new work in the area of volume management.

1 Introduction

Managing large amounts of storage can be a difficult and sensitive task. Good management tools and infrastructure will allow more reliable storage systems to be deployed and administered. Volume managers have long been a cornerstone of storage subsystems. Specific features of current Linux volume managers will be discussed in detail with a focus on how to use them. After a review of these current technologies, new developments in the field will be reviewed.

1.1 What is a Volume Manager?

Large file systems require the capacity of several disks, but most file systems must be created on a single device. A hardware RAID device is one solution to this problem. A hardware RAID device appears as a single device while in fact containing several disk drives internally. There are other excellent benefits of hardware RAID, but it is an expensive solution if one simply needs to make many small disks look like a single big disk. Volume managers are the software solution to this problem. A volume manager is typically a mid-level block device driver (often called a volume driver) which makes many disks appear as a single logical disk [Vahalia]. In addition to existing in the kernel's block I/O path, a volume manager requires user level programs to configure and manage partitions and volumes. The virtualized storage perspective produced by volume managers is so useful that often all storage, including hardware RAID, is controlled with a volume manager.

1.2 Linux Landscape

The Linux software RAID driver and the Linux Logical Volume Manager (LVM) are the two primary volume managers being used in Linux. This introduction will set the stage for further detailed descriptions of their features and how to use them in later sections.

At the end of the paper the reasons to choose one or the other should be clear.

In the simplest case a volume driver will logically append several disks one after the other as shown in Figure 1. For negligible additional cost, the driver can interleave the logical blocks among the lower level disks. Often called striping or software RAID-0, this technique provides no additional redundancy, but significantly increases the bandwidth of the logical device. In the same manner, the volume driver can implement higher RAID levels for increased redundancy. The Linux software RAID driver implements RAID levels 0, 1, and 5 [Vadala].

The management interfaces presented to the user is one emphasis of the Linux LVM [Mauelshagen]. LVM extends the concept of a single logical disk and provides *multiple* logical levels from which to manage storage. The additional abstraction levels allow logical *groupings* of storage to be manipulated independent of the actual logical devices which are being used. The user gains flexibility in allocating, moving, and replacing specific devices. Managing large and dynamic collections of storage hardware becomes much easier.

The latest feature of the Linux LVM is snapshotting. Snapshots provide an efficient way to backup an image of any file system built on an LVM logical volume.

2 Linux Kernel Interfaces

This section describes how volume managers are linked into the block device driver layer of the Linux 2.4 kernel. Those less interested in the kernel functions of volume managers may want to skip to the next section.

All read and write requests which enter the block I/O layer are broken into fundamental operations described by buffer_heads. These buffer_heads are the basic I/O descriptors used by device drivers. It is worth noting that buffer_heads are no longer a significant caching mechanism. The page cache is now the primary caching location. Buffer_heads are still used, however, to describe I/O transfers to the page cache.

Once buffer_heads have been created to describe a particular I/O operation, they are passed into the functions submit_bh() or ll_rw_block() to be queued for a mid-level driver. The specific mid-level driver (e.g. SCSI disk driver) is selected using the buffer_head's "rdev" field which is the device number. The major number of rdev is used as an offset into a table of block device drivers. When a volume manager is used, the buffer_head's rdev and rsector values specify a block location at the logical level. At the end of the process explained in the next section, the buffer's rdev and rsector values will be translated to describe a real physical block.

The mid-level driver periodically removes requests which have been placed on its queue and sends them to lower level host bus adapter drivers. The next section describes in more detail how requests are formed.

2.1 Block I/O Request Path

Each mid-level driver registered with the block device layer is defined in the blk_dev[] table by a blk_dev_struct. This structure provides a default request_queue for the driver as well as a method by which the driver can provide an alternate request_queue. The request_queue structure contains a number of important items. First, the head of the list of I/O requests for the driver is found here. Requests are taken from this list (and sent to low level drivers) after they've accumulated and some merging has been done with them. The functions used to merge and dispatch the queued requests are linked into the request_queue structure as well.

Between a buffer_head (bh) entering the block I/O path and a request being dispatched from the mid-level driver's queue, the bh must be transformed into a "request structure" which is then added to the driver's queue. To do these operations, the driver's request_queue structure supplies a function called make_request_fn.

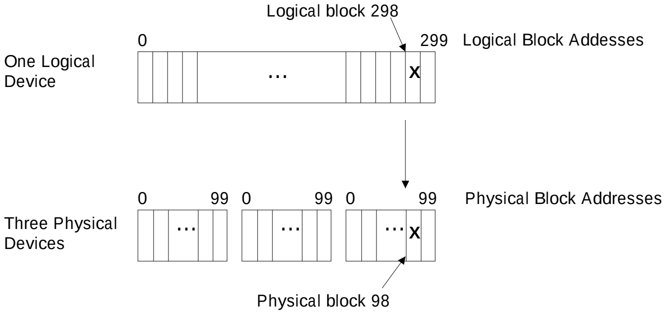

Figure 1: Linear Volume Mapping

As stated, make_request_fn will normally transform the bh into a request structure and then add the request to the driver's queue. In the case where the block driver is a volume manager, the make_request_fn simply rewrites the rdev and rsector fields of the bh to describe a real physical device and sector number. The function then returns a positive value indicating to the block I/O layer that the bh should be reprocessed. When the bh is reprocessed, a different driver with a new make_request_fn method is selected. This switch happens simply because the major number of the bh's rdev changed.

The final make_request_fn routine will behave normally and create a request structure from the bh which is added to the request queue of the mid-level driver. Volume managers can be stacked on one another in which case the bh will be reprocessed multiple times.

The Linux 2.2 kernel provides no way for volume drivers to define their own methods. The old software RAID driver is called explicitly in the block I/O path when it is being used. The Linux LVM 2.2 kernel patches from Sistina Software add a generic mechanism to the block I/O path similar to the 2.4 kernel approach for volume managers to be inserted.

3 Linux LVM

Constructing flexible virtual views of storage with LVM is possible because of the well-defined abstraction layers. User space tools used to configure each virtual level follow a similar set of operations. With these tools, online allocation or deallocation of storage to virtual groups is possible. Higher level virtual devices can then be expanded or shrunk online. Ultimately, a file system resizer is likely to be used.

The lowest level in the LVM storage hierarchy is the Physical Volume (PV). A PV is a single device or partition and is created with the command: `pvcreate /dev/sdc1`. This step initializes a partition for later use. On each PV, a Volume Group Descriptor Area (VGDA) is allocated to contain the specific configuration information. This VGDA is also kept in the `/etc/lvmtab.d` directory for performance reasons. The library functions of the LVM tools can access the local VGDA copy in a more time efficient manner than accessing all of the PVs.

Multiple Physical Volumes (initialized partitions) are merged into a Volume Group (VG). This is done with the command: `vgcreate test_vg /dev/sdc1 /dev/sde1`. Both sdc1 and sde1 must be PV's. The VG named `test_vg` now has the combined capacity of the two PV's and more PV's can be added at any time. This step also registers `test_vg` in the LVM kernel module and therefore it is made accessible to the kernel I/O layer.

A Volume Group can be viewed as a large pool of storage from which Logical Volumes (LV) can be allocated. LV's are actual block devices on which file systems or databases can be created. Creating an LV is done by: `lvcreate -L1500 -ntest_lv test_vg`. This command creates a 1500 MB linear LV named `test_lv`. The device node is `/dev/test_vg/test_lv`. Every LV allocated from `test_vg` will be in the `/dev/test_vg/` directory. Other options can be used to specify an LV which is striped or has a physically contiguous allocation policy.

A Volume Group (VG) splits each of its underlying Physical Volumes (PV's) into smaller units of Physical Extents (PE's). One PE is typically a few megabytes but can range up to a few gigabytes to enable large disk subsystems. These PE's are then allocated to Logical Volumes (LV's) in units called Logical Extents (LE's). There is a one to one mapping between a LE and a lower level PE; they refer to the same chunk of space. A LE is simply a PE which has been allocated to a Logical Volume's address space.

A file system on `test_lv` may need to be extended. To do this the LV and possibly the underlying VG need to be extended. More space may be added to a VG by adding PV's with the command: `vgextend test_vg /dev/sdf1` where sdf1 has been "pvcreated". The device `test_lv` can then be extended by 100 MB with the command: `lvextend -L+100 /dev/test_vg/test_lv` if there is 100 MB of space available in `test_vg`. The similar commands `vgreduce` and `lvreduce` can be used to shrink VG's and LV's.

When shrinking volumes or replacing physical devices, all LE's must sometimes be moved off a specific device. The command `pvmove` can be used in several ways to move any LE's off a device to available PE's elsewhere. There are also many more commands to rename, remove, split, merge, activate, deactivate and get extended information about current PV's, VG's and LV's.

3.1 Practical LVM Usage

There are many different situations in which LVM can be beneficial. Two different practical examples will be described in this section; one for small and the other for large systems.

3.1.1 Flexible Space Allocation

LVM can be very useful on small single disk systems and even laptops [Sistina]. Consider, for example, a system installation which is split among three file systems mounted at /home, /usr, and root (/). If these three file systems are placed on three fixed partitions of the disk, a problem such as the following is common. The user or users of the system need much more space in their home directories than anticipated and the the /home file system becomes full. No additional software has been installed on the system so the /usr file system has plenty of extra space. The fixed partitions prevent the /home file system from being expanded and the /usr file system from being shrunk.

The problem of file system space allocation could be solved easily if LVM was used. Using LVM, a single partition would be created on the disk and this partition would be made into a single PV. A volume group named *mainvol* would then be created from the PV. From *mainvol*, three LV's would be allocated, one for each file system. All the space need not be allocated from *mainvol* immediately, but can be saved and added later to the LV of the file system which needs space first. If all space from *mainvol* has been allocated to LV's, the allocation can be adjusted. To do this, one would shrink the file system which is underutilized, lvreduce the corresponding LV, lvextend the LV needing space, and finally expand the file system on the extended LV.

If a second disk was added to the system, the new disk could be added to the VG and used for growth in any of the file systems, or new LV's could be created for new file systems.

3.1.2 Volume Groups

In large systems with many storage devices the ability to dynamically adjust volume sizes as previously described is even more important. There are also other ways in which LVM can improve administration in large installations. Volume groups can be set up for distinct applications, business divisions, or other categories. This separation is useful because storage devices are often meant for a particular purpose.

Consider the following situation in which grouping storage devices into volume groups is important because of the different applications. A single machine is managing storage for both a mail server and a database server. The mail application requires two LV's for two file systems. The two LV's are allocated from a *mailvol* VG which is a pool of several individual SCSI disks. Disks are added and replaced in *mailvol* as demand changes. The machine also exports critical business databases on two LV's. These logical volumes are allocated from the *datavol* VG which is composed of two hardware RAID devices. It would be inappropriate for mail file systems to use space on the RAID devices, or for database logical volumes to use space on the individual SCSI disks. Separate VG's make it easy to ensure that an application will use storage space on the correct hardware.

3.1.3 Maintenance Tasks

LVM makes it simple to replace slow, small or aging disks in a VG while the LV's and file system's remain on-line. The pvmove command will move all logical extents off of a specified disk to free locations in the volume group on other devices. There are limitations to this when LV's are striped as well as problems if the system crashes during the pvmove operation which can potentially take a long time.

3.2 Logical Volume Snapshots

Snapshots are "frozen" images of a logical volume which can be used to back up the resident file system or database. A user command is used to initiate a snapshot at which point the current state of the LV is preserved and can be accessed through a special device node corresponding to the active LV's device node. A backup process uses the read-only "snapped" LV which does not change. The file system or database can continue running while the backup takes place.

LVM uses a copy-on-write technique to allow continued updates to an LV while maintaining a previous frozen state. Any new writes result in a copy of the original block into a separate volume before the changed block can be written. The old and new locations of each modified chunk are maintained by LVM. Pointers to the original data are used when the snapshotted volume is accessed. The metadata managing copy-on-write mappings is persistent so snapshots remain available after reboots.

A snapshot LV requires only a fraction of the space of the original LV because only changed data needs new space. The actual size should be tuned according to the frequency of writes to the live volume and the life span of the snapshot volume. If write activity changes to the original volume after the snapshot has been taken, the snapshot size can be changed at run time. This is especially useful if too little space was allocated for the snapshot volume.

A snapshot is created as a new LV in the same volume group as the target LV. The volume group must have free space available to allocate the snapshot LV. The name of the snapshot LV can be assigned by the user just as the name of any new LV.

3.2.1 Application Consistency

Taking snapshots at a logical volume level is problematic if the file system, database, or user application is not in a consistent state at the moment the snapshot is taken. There are a variety of approaches to solving this problem depending on the particular application.

Databases often have hooks which can be used to quiesce them momentarily while a snapshot is taken. This will guarantee a consistent database state in the snapshot. A separate database specific program would likely coordinate the necessary database operations before and after the LVM snapshot is taken.

If a snapshot is simply taken of a file system volume, the resulting copy of the file system will look as it would if the system had crashed. When the snapped file system is then mounted read-only, it would appear to need an fsck in the case of a non-journalled file system or journal recovery in the case of a journalled file system. These operations (fsck or journal replay) require writing to the snapshot logical volume which is not allowed.

One solution to this problem is to unmount the file system before taking the snapshot and then remount the file system afterward. Requiring the file system or database to be taken off-line is not a widely accepted solution, however. Another approach would be to implement writable snapshots so recovery could be done when the snapshot file system is mounted. A third solution is to create a file system interface which a snapshot procedure could invoke to cause the file system to make its on-disk state consistent. In this case, recovery would not be needed.

Adding file system hooks for snapshots is the method being pursued and a kernel patch adds initial support for it. This approach still fails to address the issue of user mode applications which can be caught in an inconsistent state despite the file system itself being consistent. The snapshot could be taken between two writes from an application, both of which are required for the application to be consistent. This is a difficult problem because there are no standard API's to tell applications to make themselves consistent.

4 Software RAID

RAID (Redundant Array of Independent Disks) is a set of methods (or levels) for storing data on a set of disks to improve performance, reliability, or both [Patterson]. RAID levels 0, 1 and 5 are most common. RAID is often implemented in hardware controllers. A RAID controller appears to a host computer just as any other storage device would. Behind the hardware RAID controller is a group of disk drives. Depending on which RAID level the controller is configured for, it will store data on the disks in different ways. The common RAID levels have the following characteristics:

- *RAID-0* improves performance by striping data blocks across the individual disks. Because all drives are ideally being used in parallel, the overall performance is the combination of the performance of each disk. Striping also reduces the chance that one disk will be subject to much more load than others (a "hot-spot"). There is no reliability or redundancy added by this level.
- *RAID-1* is also known as mirroring. The total usable storage space is half the physical storage capacity because half the disks are mirrored on the other half. Reliability is maximized because every disk has a duplicate. If a disk fails, its mirrored copy is synced to a spare or replacement disk. The device continues operating during recovery.
- *RAID-5* increases reliability allowing any one disk to fail without losing information. The space overhead is also low. Exclusive OR or "parity" values are calculated for each block stripe within the disk group. Blocks containing the parity values are interleaved among all the disks. If one disk fails, the data which was on that disk can be reconstructed using the data and parity blocks from the operational disks.

The RAID levels can also be implemented in the host's software on any collection of individual disks. RAID-0 (striping) and RAID-1 (mirroring) are the simplest to implement in a host driver and many volume drivers only support these two levels. Recovery procedures are the most difficult aspect of software RAID.

Performance of software RAID may be slower than hardware RAID for a couple reasons. Software RAID levels one and higher often require more data to be transfered between hosts and storage than would be required for hardware RAID. For example, the host must make two writes to separate disks when maintaining mirrors whereas the data would be written only once to a hardware RAID device. In addition to the extra I/O to maintain parity, RAID-5 XOR calculations require extra CPU cycles.

One significant limitation in many software RAID drivers is the inability to extend RAID-0 (striped) or RAID-5 volumes. Data placement by either of these methods often is dependent directly on the total number of devices in the RAID set. Adding one additional disk would require shifting the location of all current data.

RAID-0 and RAID-1 are often combined to gain the advantages of both. They can be combined as RAID-0+1 which means that two volumes are mirrored while each volume is striped internally. The other combination is RAID-1+0 where each disk is mirrored and striping is done across all mirrors. RAID-0+1 may be considered less reliable because if each half looses any one disk, the entire volume fails.

4.1 Linux MD Driver Usage

The MD (Multiple Devices) software RAID volume driver has been entirely rewritten and vastly enhanced between the 2.2 and 2.4 kernels [Molnar]. In terms of logical devices, the MD driver provides the basic logical device abstraction without the management or snapshot capabilities of LVM. The focus of MD is high performance RAID and efficient background reconstruction in the event of disk failures. If extra devices are installed beforehand, they may be used as "hot-swappable" replacements by the driver.

There is much less administration in comparison to LVM because of the single device abstraction and the fact that most MD features are automatic. The primary step in creating an MD device is creating the logical definition in the /etc/raidtab file. Figure 2 shows a sample raidtab file. The contents of the file are:

```
raiddev /dev/md0
        raid-level      0
        nr-raid-disks   8
        persistent-superblock 1
        chunk-size      64
        device          /dev/sdb1
        raid-disk       0
        device          /dev/sdc1
        raid-disk       1
        device          /dev/sdd1
        raid-disk       2
        device          /dev/sde1
        raid-disk       3
        device          /dev/sdf1
        raid-disk       4
        device          /dev/sdg1
        raid-disk       5
        device          /dev/sdh1
        raid-disk       6
        device          /dev/sdi1
        raid-disk       7
```

Figure 2: A sample /etc/raidtab file.

- *raiddev* identifies which logical device is being described. The MD logical devices are represented by /dev/md0, /dev/md1, etc.
- *raid-level* determines the level of software RAID MD will do for this device. The levels are 0 for striping, 1 for mirroring, 5 for parity RAID, or "linear" for no RAID.
- *nr-raid-disks* defines the number of disks within the logical device.
- *persistent-superblock* if set to "1" will make MD store configuration details (called a "superblock") on each device. This allows auto-detection of MD devices by the kernel at boot time.
- *chunk-size* defines the stripe width when using RAID-0. It is the number of 1 KB units within a chunk, so 64 means data is interleaved at 64 KB boundaries.
- *device* specifies the device node of a partition assigned to the logical volume. There are as many *device* lines as *nr-raid-disks*.
- *raid-disk* follows every *device* line to indicate which sub-disk of the logical volume the *device* should be.
- *spare-disk* is used instead of *raid-disk* if the *device* is to be used as a replacement for any disk which fails.

The first time an MD device is set up, the command `mkraid` is used. The specific MD device (e.g. /dev/md0) follows the command on the command line. The raidtab file is read to find the device definition. A configured MD device can be enabled or disabled using the commands `raidstart /dev/md0` and `raidstop /dev/md0` [Østergaard].

5 GFS's Pool Driver

The Pool driver [Teigland] is a simple Linux volume manager which has been developed in conjunction with the Global File System [Preslan]. Three features of the Pool volume manager motivated by needs of GFS are mentioned here. In the future, it would be beneficial to integrate some of these features into another more widely used volume manager like LVM.

5.1 SCSI Mid-layer Interface

GFS is a shared disk cluster file system for Linux. Nodes in a GFS cluster must use a locking mechanism for synchronization. The first unique feature of Pool exists specifically to support GFS cluster locking. GFS can use a new SCSI command called the Device Memory Export Protocol, or DMEP, (still in the process of standardization) to access a pool of memory buffers implemented in the firmware of hardware RAID controllers and possibly disk drives [Barry]. When a Pool device is configured, the constituent real devices are tagged as supporting the DMEP command or not.

The Pool driver knows the physical distribution of DMEP buffers and presents a single logical pool of them to a GFS lock module (or other application). Primitive DMEP buffer operations like load and conditional store can be used to implement cluster wide locks [1].

The Pool driver maps a logical DMEP buffer reference to a specific DMEP buffer on a real device and then sends the DMEP command to a SCSI target. To send the DMEP command, the Pool driver must bypass the standard SCSI driver APIs in the kernel designed for I/O requests and insert the DMEP Command Descriptor Block (CDB) in the SCSI command queue.

It would be possible to use GFS above a different cluster volume manager and continue to use Pool to access DMEP buffers. No regular data blocks would ever be read from or written to the Pool device and the size of the Pool could be zero. This would not require any changes to Pool and the cluster volume manager would not need any hooks for DMEP processing. This simple arrangement would be accomplished by creating a small partition on each DMEP capable device. These partitions would make up the Pool. The program which creates Pools writes a small label to the head of each Pool device, so the partitions must only be large enough to hold a Pool label.

5.2 Device Identity

The second feature of Pool is based on the knowledge that GFS and Pool will be used in a storage area network (SAN) or other environment where devices are shared directly among multiple hosts. A problem in SAN's are devices which change "identity" when the Fibre Channel loop or fabric is re-initialized. The volume manager can not assume that a physical device will always be matched with the same target address or device node.

To solve this problem, the Pool driver identifies Pool partitions by a label placed at the head of each Pool partition. At startup, Pool scans all the devices for specific Pool labels and when all the partitions have been found, the Pool can be activated. The labels impose a fixed ordering on the Pool partitions with respect to each other, but with no respect to physical ID.

[1] GFS can use other distributed locking methods as well.

5.3 Sub-Pools

The third feature of Pool is the ability to configure a Pool device with distinct “sub-pools” consisting of particular partitions or LUNs. In addition to common Pool device characteristics (like total size), the file system or mkfs program can obtain sub-pool parameters and use sub-pools in special ways. Currently, a GFS host can use a separate sub-pool for its private journal while the ordinary file system data is located in general “data” sub-pools. This can improve performance if a journal is placed on a separate device.

Performance can be further optimized because disk striping is configured per sub-pool. If many disks are used in a logical volume, striping among subsets of disks can be beneficial due to improved parallelism among sub-pools (e.g. four sets of four striped disks rather than 16 striped disks). Other possible uses for sub-pools or similar constructions in other volume managers are covered later.

6 Volume Managers on other Platforms

Volume managers have appeared in many different UNIX systems in many forms. This section will review a few of the major ones.

6.1 Veritas Volume Manager

The Veritas Volume Manager (VxVM) is a very advanced product which supports levels of configuration comparable to Linux LVM as well as the RAID levels of Linux software RAID [Veritas]. VxVM runs on HP-UX and Solaris platforms. Java graphical management tools are available in addition to the command line tools.

The abstraction layers of VxVM are more complex than those found in Linux LVM. The abstract objects include: Physical Disks, VM Disks, Subdisks, Plexes, Volumes, and Disk Groups [Veritas SAG].

The VxVM RAID-0 (striping) implementation allows a striped plex to be expanded. This is possible because of the way VxVM virtual objects are constructed. RAID-0 can be used with RAID-1 to combine the advantages of both.

The RAID-1 (mirroring) capabilities can be tuned for optimal performance. Both copies of the data in a mirror are used for reading and adjustable algorithms decide which half of the mirror to read from. Writes to the mirror happen in parallel so the performance slowdown is minimal. There are three methods which VxVM can use for mirror reconstruction depending on the type of failure. Dirty Region Logging is one method which can help the mirror resynchronization process.

The RAID-5 implementation uses logging which prevents data corruption in the case of both a system failure and a disk failure. The write-ahead log should be kept on a solid state disk (SSD) or in NVRAM for performance reasons. Optimizations are also made to improve RAID-5 write performance in general.

The VxVM “Hot-Relocation” feature is very similar to using hot-spares in Linux software RAID.

6.2 Sun Solstice DiskSuite

Sun’s Solstice DiskSuite (SDS) is a volume manager which runs on the Solaris operating system only [Sun]. SDS supports most of the VxVM features but is generally considered less robust than VxVM. The Metadisk is the only abstract object used by SDS limiting the configuration options significantly. The following list gives a basic comparison with VxVM.

- RAID-0 stripe sets cannot be expanded, although other stripe sets can be concatenated with existing sets to expand a volume containing striped devices.
- Proper RAID-1 mirror resynchronization is supported much like VxVM.

- Hot Spare disks can be configured.
- RAID-5 sets are supported, but can only be extended in the same limited fashion as stripe sets. RAID-5 reconstruction is a manual operation and does not include logging to guarantee data recovery. SDS does not include RAID-5 optimizations to improve write performance.

Both SDS and VxVM provide some level of support for coordinating access to shared devices on a storage network. This involves some control over which hosts can see, manage or use shared devices.

6.3 FreeBSD Vinum

Vinum is a volume manager implemented under FreeBSD and is Open Source like the Linux volume managers [Lehey]. Vinum implements a simplified set of the Veritas abstraction objects. In particular Vinum defines: a *drive* which is a device or partition (slice) as seen by the operating system; a *subdisk* which is a contiguous range of blocks on a drive (not including Vinum metadata); a *plex* which is a group of subdisks; and a *volume* which is a group of one or more plexes. Volumes are objects on which file systems are created.

Within a plex, the subdisks can be concatenated, striped, or made into a RAID-5 set. Only concatenated plexes can be expanded. Mirrors are created by adding two plexes of the same size to a volume. Other advanced features found in the previous commercial products are not currently supported but are planned as a part of future development.

6.4 Linux LVM Relatives

IBM initially developed an LVM which was subsequently adopted by the OSF (now OpenGroup) for their OSF/1 operating system. The OSF version was then used as a base for the HP-UX and Digital UNIX operating system LVM implementations. The abstraction model in this family of LVM's is different from the Veritas model. The Linux LVM implementation is similar to the HP-UX LVM implementation.

7 New Work in Linux

This section covers some more specialized topics and lists some areas of current and future research in Linux volume management.

7.1 Exporting Volume Metadata

The support for sub-pools in the Pool volume manager has already been mentioned. The general advantage of sub-pools is the ability to specify *sections* of a logical volume as having particular characteristics. The file system or database using the logical volume could take advantage of information (or metadata) obtained from the volume driver which matches qualitative information with block ranges. The information could be set permanently during configuration or it could represent statistical information gathered by the volume driver.

In the case of GFS, performance can be improved by simply placing different types of data in sections of the logical volume which map to different physical devices. This doesn't require any additional metadata than what a volume manager already uses. It simply requires a method of exporting this metadata to systems which can take advantage of it. If more detailed device characteristics were maintained by the volume driver, like I/O rates or reliability, the file system could be tuned even further.

One approach being considered for Linux LVM is adding an API for reading and writing exported chunks of metadata. The content of these chunks would be opaque to LVM itself. A file system or other application could use this exposed metadata for a variety of purposes including those already mentioned. The first difficulty is determining what volume geometry to export along with the metadata. Volume managers will have varying degrees of difficulty exposing useful volume geometry depending on the internal abstraction layers.

7.2 Cluster LVM

A Cluster LVM will coordinate operations in a shared storage environment. The future extensions of LVM will aim to implement a fully cluster transparent use of the existing command line interfaces and features. This will enable a user to run the familiar Linux LVM commands on an arbitrary cluster node.

In the first Cluster LVM release, a single global lock will allow only one LVM command to be run in the cluster at a specific point in time so that LVM metadata (VGDA's) are updated correctly. Cached metadata on all cluster nodes will also be made consistent. The global lock will be supported by a simple synchronous distributed locking API.

Later releases of the Cluster LVM will use a more fine grained locking model. This will allow pieces of the VGDA to be locked and updated individually. One of the advantages of fine grained locking is that the amount of metadata updated on all cluster nodes is minimized which reduces the latency of operations. Another advantage is that commands dealing with different LVM resources can run in parallel (e.g. creating two LV's in different VG's).

Some volume groups and logical volumes on the shared storage may be intended for a single node only. Other volumes may be for concurrent use as is the case when using a cluster file system like GFS. Metadata enhancements will be added to support different sharing modes on volumes.

7.3 Combining Volume Types

MD RAID volumes can be used along with LVM to gain the advantages of both volume managers. An MD device can be used as a Physical Volume within an LVM system. In this case, whatever data is placed by LVM on the MD device will be more reliable as MD implements the RAID functions independently of LVM. The ordinary LVM tools can be used to manage the storage devices although the partitions within the MD volume could not be managed individually by LVM.

7.4 IBM EVMS

IBM's Enterprise Volume Management System (EVMS) is a volume manager project with the aim of producing a highly flexible system which could emulate the technical features and operational interfaces of nearly any other volume manager [Rafanello]. Each of the following abstraction layers in the EVMS architecture accepts "pluggable" modules which can deliver different features or interact with differing system types.

- *Device Managers* interact with the kernel's block device drivers to obtain basic information about what physical devices are connected to the machine. The bottom part of this layer must be very operating system and even kernel version specific. Device Managers present abstracted Logical Disks to the layer above.

- *Partition Managers* recognize specific partition types on Logical Disks. Rather than enforcing its own partition type on a disk, this layer allows Partition Managers to translate various disk partitions into abstract "Logical Partitions" which are used by the next layer.

- *Volume Group Emulators* merge Logical Partitions into Volume Groups. The Volume Group concept is slightly different on different platforms, so modules at this level create the behavior for the specific system of interest (Linux LVM, AIX LVM). Partition based volume managers which don't have the volume group concept can presumably be emulated as well.

- *Features* create usable Logical Volumes from Volume Groups. Different Feature modules will map and translate blocks differently between Logical Volumes and

Volume Groups. This is where software RAID can be performed or other operations like encryption. Feature modules can be layered above one another to combine their effects. Features are categorized or layered according to whether they operate at a volume level (e.g. encryption), at the aggregation level (e.g. software RAID), or at a partition level (e.g. bad block relocation).

- *File System Interface Modules* are not layers in the I/O stack like the others. Modules plug in here to interface with different file systems for the purpose of resize operations, or other things which the volume manager and file system need to communicate about.

8 Conclusion

This paper has described the current Linux volume managers and has reviewed the ongoing work to extend and improve them. Today, both the Linux LVM and Linux software RAID drivers are widely used. Those interested in maximal flexibility and manageability often use Linux LVM, especially when there are large numbers of devices to manage or large numbers of volumes to maintain. Those interested in high availability without the high cost of special hardware often use Linux software RAID in mirrored or RAID-5 configurations. Both products will continue to improve and simple methods to use them both together will likely emerge.

9 Acknowledgments

The authors would like to acknowledge the LVM group at Sistina Software for their input on this paper and the work on LVM. This includes: Patrick Caulfield, AJ Lewis, and Joe Thornber. We would also like to thank Matthew O'Keefe of Sistina for his input on this paper.

References

[Barry] Andrew Barry, Kenneth Preslan, Matthew O'Keefe SCSI Device Memory Export Protocol, `http://www.sistina.com/gfs/Pages/dmep.html`

[Lehey] Greg Lehey *The Vinum Volume Manager*, `http://www.vinumvm.org`

[Mauelshagen] Heinz Mauelshagen, Primary author and maintainer of Linux Logical Volume Manager, `http://www.sistina.com/lvm`

[Molnar] Ingo Molnar, Primary author and maintainer of Linux software RAID, `http://people.redhat.com/mingo/`

[Østergaard] Jakob Østergaard, *The Software-RAID HOWTO* `http://www.linuxdoc.org/HOWTO/Software-RAID-HOWTO.html`.

[Patterson] D. Patterson, G. Gibson, R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of the 1988 ACM SIGMOD Conference of Management of Data*, June 1988.

[Preslan] Kenneth Preslan, Andrew Barry, Jonathan Brassow, Russell Cattlen, Andam Manthei, Erling Nygaard, Seth Van Oort, David Teigland, Mike Tilstra, Matthew O'Keefe, "Implementing Journaling in a Linux Shared Disk File System", *Proceedings of the Eighth NASA Goddard Conference on Mass Storage Systems and Technologies in cooperation with the Seventeenth IEEE Symposium on Mass Storage Systems*, March 2000.

[Rafanello] Ben Rafanello, John Stiles, Cuong Tran, *Emulating Multiple Logical Volume Management Systems within a Single Volume Management Architecture* `http://oss.software.ibm.com/ developerworks/ opensource/ evms/ doc/ EVMS_White_Paper_1.htm`

[Sistina] *LVM HOWTO*, `http://www.sistina.com/ lvm/doc/ lvm_howto/index.html`

[Sun] *Solstice DiskSuite 4.0 Administration Guide*, `http://docs.sun.com`

[Teigland] David Teigland "The Pool Driver: A Volume Driver for SANs," Master's thesis, Dept. of Electrical and Computer Engineering, Univ. of Minnesota, Minneapolis, MN, Dec. 1999.

[Vadala] Derek Vadala "RAID on Linux", *Journal of Linux Technology*, Vol. 1, No. 2, April 2000, pp. 25-33.

[Vahalia] Uresh Vahalia, *UNIX Internals*, Prentice Hall, 1996.

[Veritas] *Features of VERITAS Volume Manager for Unix and VERITAS File System*, `http://www.veritas.com/us/products/ volumemanager/whitepaper-02.html`

[Veritas SAG] *VERITAS Volume Manager System Administrator's Guide*, `http://uw7doc.sco.com/ODM_VMadmin/sag-1.html`

Linux LVM, Pool and GFS publications, HOWTOs and source code can be found at `http://www.sistina.com`.