Design of a Cluster Logical Volume Manager

Abhinay Ramesh Kampasi
Pune Institute of Computer Technology,
University of Pune.
Email: *abhinaykampasi@hotmail.com* ***WWW:*** *http://abhinaykampasi.tripod.com*

CERTIFICATE

This is to certify that

Mr. Abhinay Ramesh Kampasi

has completed the necessary seminar work and prepared the bonafied report on

Design of a Cluster Logical Volume Manager

in a satisfactory manner as partial fulfillment for requirement of the degree of
B.E (Computer) at
University of Pune in the academic year 2002-2003

Date: 14th August 2002

Place: PICT, Pune

Prof. Dr. C V K Rao	Prof. G P Potdar	Prof. Dr. C V K Rao
Internal Guide	Seminar coordinator	H.O.D

DEPARTMENT OF COMPUTER ENGINEERING
PUNE INSTITUTE OF COMPUTER TECHNOLOGY
PUNE - 43

Acknowledgements

On completion of this report I would like to thank my teachers, advisors, friends and well wishers without whom this report would not have been possible. I am thankful to Dr. Prof. CVK Rao and Prof. GP Potdar for their support.

I take this opportunity to mention specifically the contributions of **Mr. Adityashankar Kini** at Calsoft Inc. who started me out on the topic. He guided me to understand the intricacies involved in this design and provided extremely useful review feedback.

Abhinay Ramesh Kampasi

Table of Contents

Design of a Cluster Logical Volume Manager

Abhinay Ramesh Kampasi
Pune Institute of Computer Technology
University of Pune
E-mail:abhinaykampasi@hotmail.com **WWW**:http://abhinaykampasi.tripod.com

Abstract:

Logical Volume Managers provide a higher level view of disk storage on a computer system than the traditional view of disks and partitions. However the functionality of the existing logical volume managers is restricted to a single machine and they cannot be easily incorporated in a cluster environment. This is because logical volume managers today do not support distributed updates and locking on metadata. I propose a design of a Cluster Logical Volume Manager which incorporates a distributed locking mechanism on metadata to enforce a single node exclusive access to cluster metadata. The design can be incorporated in practically all the logical volume managers available today. As a case study, I have proposed a design for the Cluster Logical Volume Manager, which enhances the Linux Logical Volume Manager to function in a cluster. The design incorporates a cluster lock manager in the architecture, which ensures that the metadata operations are handled very carefully ensuring stability of the cluster data.

1 Logical Volume manager

1.1 What is a Logical Volume Manager?

In an open systems environment, logical volume manager (LVM) virtualizes storage by consolidating physical storage into logical volumes, which are available to applications or file systems. LVMs are available on most Unix platforms (including Linux) and on Windows 2000. By assembling virtual storage volumes out of numerous physical devices, you can create storage configurations tuned for specific applications, without the limitations of specific hardware devices. Instead, you can make use of the available storage, without locking into proprietary storage solutions. Logical volume managers improve application availability by building redundancy into the logical volume itself. The possibilities go beyond simple mirroring and RAID. For example, the failure of a device in a redundant storage configuration can degrade performance and expose the data to risk from another failure. The logical volume manager can maintain a pool of spare disks that can be automatically swapped in (hot relocation) when a device fails in a logical volume. It can even move data back automatically when the failed device is replaced or repaired. Unlike a physical device, a logical volume has a nearly limitless capacity: Administrators can add storage as needed without interrupting access to the data. When used with a database's auto-extend capabilities or a file system's automatic extension, a logical volume manager can significantly ease the problem of provisioning storage for growing applications.

LVM supports enterprise level volume management of disk and disk subsystems by grouping arbitrary disks into volume groups. The total capacity of volume groups can be allocated to logical volumes, which are accessed as regular block devices. Further, LVM provides logical separation of storage, the ability to move data from one physical device to another while on-line, and dynamic block device resizing. LVM also enables system administrators to upgrade systems, remove failing disks, reorganize workloads, and adapt to changing system needs, through a minimum amount of time and effort. LVM is a major building block for the Linux OS because it allows Linux to be used in an enterprise server environment. In addition to addressing the needs of the enterprise computing sector, LVM provides ease-of-use for desktop users as well, by simplifying on-disk management of Linux file systems and volumes.

1.2 What is Volume Management?

Volume Management creates a layer of abstraction over the storage. Applications use a virtual storage, which is managed using an LVM [7]. Volume management lets you edit the storage configuration without actually changing anything on the hardware side, and vice versa. By hiding the hardware details it completely separates hardware and software storage management, so that it is possible to change the hardware side without the software ever noticing, all during runtime. With an LVM system uptime can be enhanced significantly, because for changes in the storage configuration no application has to be stopped any more.

Logical volume management provides a higher-level view of the disk storage on a computer system than the traditional view of disks and partitions. This gives the system administrator much more flexibility in allocating storage to applications and users. Storage volumes created under the control of the logical volume manager can be resized and moved around almost at will, although this may need some upgrading of file system tools.

1.3 Use of a Logical Volume Manager:

Logical volume management is traditionally associated with large installations containing many disks but it is equally suited to small systems with a single disk or maybe two. One of the difficult decisions facing a new user installing Linux for the first time is how to partition the disk drive. The need to estimate just how much space is likely to be needed for system files and user files makes the installation more complex than is necessary and some users simply opt to put all their data into one large partition in an attempt to avoid the issue. Once the user has guessed how much space is needed for /home /usr / (or has let the installation program do it) then is quite common for one of these partitions to fill up even if there is plenty of disk space in one of the other partitions.

With logical volume management, the whole disk would be allocated to a single volume group and logical volumes created to hold the / /usr and /home file systems. If, for example the /home logical volume later filled up but there was still space available on /usr then it would be possible to shrink /usr by a few megabytes and reallocate that space to /home. Another alternative would be to allocate minimal amounts of space for each logical volume and leave some of the disk unallocated. Then, when the partitionsstart to fill up, they can be expanded as necessary.

As an example:

Consider a PC with an 8.4 Gigabyte disk on it and Linux installed on it. The computer uses the following partitioning system:

/boot	/dev/hda1	10 Megabytes
swap	/dev/hda2	256 Megabytes
/	/dev/hda3	2 Gigabytes
/home	/dev/hda4	6 Gigabytes

Now at a later stage, if the configuration is to be changed such that more space is available for the root /.

The possible ways of achieving this are as follows:

1. Reformat the disk, change the partitioning scheme and reinstall.
2. Buy a new disk and figure out some new partitioning scheme that will require the minimum of data movement.
3. Set up a symlink farm from / to /home and install the new software on /home

With LVM this becomes much easier:

Consider a similar PC, which uses LVM to divide up the disk in a similar manner:

/boot	/dev/vg00/boot	10 Megabytes

swap	/dev/vg00/swap	256 Megabytes
/	/dev/vg00/root	2 Gigabytes
/home	/dev/vg00/home	6 Gigabytes

If there is a similar problem one can reduce the size of /home by a gigabyte and add that space to the root partition. Also, if the /home partition is used up as well and a new 20 Gigabyte disk is added to both the systems.

In the first case, one has to format the whole disk as one partition (/dev/hdb1) and moves the existing /home data onto it and uses the new disk as /home. However now there is 6 gigabytes unused and one has to use symlinks to make that disk appear as an extension of /home, say /home/joe/old-mp3s. On the second machine, one can simply add the new disk to the existing volume group and extend the /home logical volume to include the new disk. Or, in fact, one could move the data from /home on the old disk to the new disk and then extend the existing root volume to cover the entire old disk

1.4 Operation of the LVM:

The LVM consists of:

• High-level commands

• Intermediate-level commands

• Logical volume manager subroutine interface library

• Logical volume device driver

• Disk device driver

• Adapter device driver

Figure 1.1 shows path from a high-level command (mklv) to an intermediate command (lcreatelv) to a library function (lvm_createlv) to the logical volume device driver to the disk device driver to the SCSI device driver and to the physical volume. Other applications, for example, the journaled file system, also talk directly to the LVDD[18].

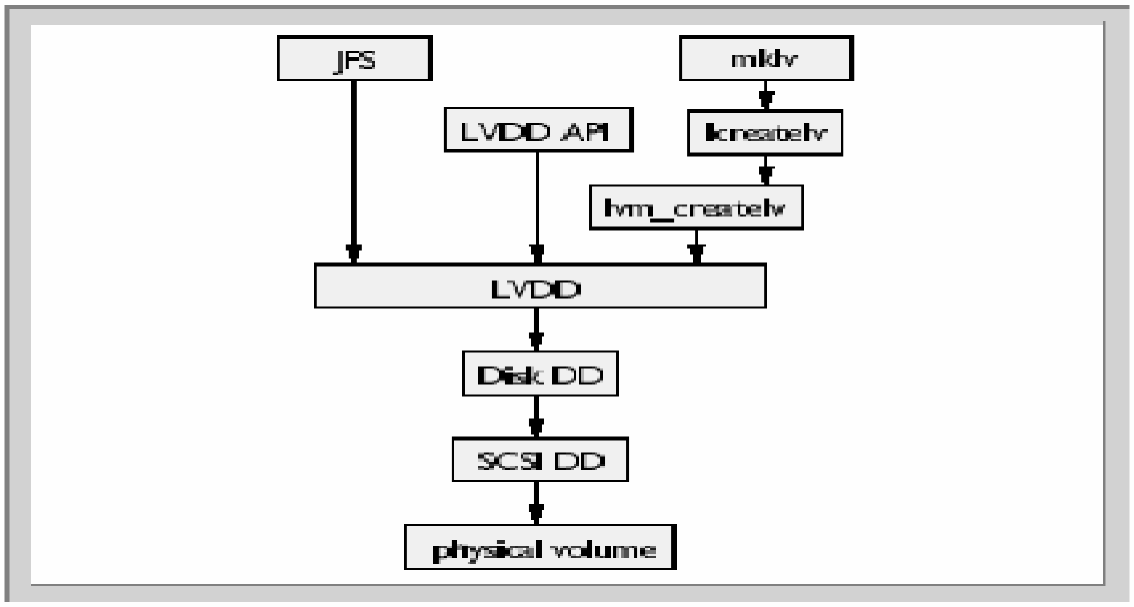


Fig. 1.1 Flow from High Level commands to Physical Volume

1.4.1 High Level Commands:

These are used by System Management Interface Tool (SMIT), experienced users and have a high-level of error checking [18]. Most of the LVM commands reside in /usr/sbin (except /etc/cfgvg), and they are a combination of shell scripts and C binaries:
Please refer Appendix C for the high level commands.

1.4.2 Intermediate Level Commands:

These are used by high-level commands and provide usage statements as well as some level of error checking. The intermediate commands found in /usr/sbin are designed to be called by the high-level commands and are all C binaries:
Please refer to Appendix C for the intermediate level commands.

1.4.3 The Logical Volume Manager subroutine interface library:

These contain routines that are used by the system management commands to perform system management tasks for the logical and physical volumes of a system. As these commands give usage statements when queried from the command line, it gives users the incorrect impression that these commands are for general use in LVM. This is not true. These commands have been "tuned" for use by the high-level shell scripts. As they are designed, only high-level shell scripts will be calling these intermediate commands. Certain error checking is not performed by the intermediate commands. The high-level commands are responsible for screening potentially dangerous errors or conditions out. The removal of the unintended usage statements has been considered, but users have asked LVM to leave them, even if they are not documented.

1.4.4 Logical volume device driver:

It is a pseudo-device driver that manages and processes all I/O and operates on logical volumes through the /dev/lv special file. It translates logical addresses into physical addresses and sends I/O requests to specific device drivers. Like the physical disk device driver, this pseudo-device driver provides character and block entry points with compatible arguments. Each volume group has an entry in the kernel device switch table. Each entry contains entry points for the device driver and a pointer to the volume group data structure. The logical volumes of a volume group are distinguished by their minor numbers.

1.4.5 Disk Device Drivers:

There are various disk device drivers in LVM. The most common disk device driver is the one for SCSI device drivers: scdisk and csdiskpin (in /usr/lib/drivers). The second most common disk device driver is most often the one for Serial Storage Architecture (SSA) disks [23].

Small Computer System Interface (SCSI) [24] disk device drivers support the SCSI fixed disk, CD-ROM (compact disk read only memory), and read/write optical (optical memory) devices. Typical fixed disk, CD-ROM, and read/write optical drive operations are implemented using the open, close, read, write, and ioctl subroutines.

SSA disk drives are represented in AIX as SSA logical disks (hdisk0, hdisk1….hdiskN) and SSA physical disks (pdisk0,pdisk1.....pdiskN). SSA RAID arrays are represented as SSA logical disks (hdisk0, hdisk1.....hdiskN). SSA logical disks represent the logical properties of the disk drive or array and can have volume groups and file systems mounted on them. SSA physical disks represent the physical properties of the disk drive.

2 Structure of a Logical Volume Manager

This section will introduce the components of the Logical Volume Manager (LVM) and how they relate to the Application and Physical Layers. The set of operating system commands, library subroutines and other tools that allow the user to establish and control logical volume storage is called the Logical Volume Manager[18]. The Logical Volume Manager controls disk resources by mapping data between a simple and flexible logical view of storage space and the actual physical disks. The Logical Volume Manager does this by using a layer of device driver code that runs above the traditional physical device

drivers. This logical view of the disk storage is provided to applications and is independent of the underlying physical disk structure. **Figure 2.1 illustrates the layout of those components.**

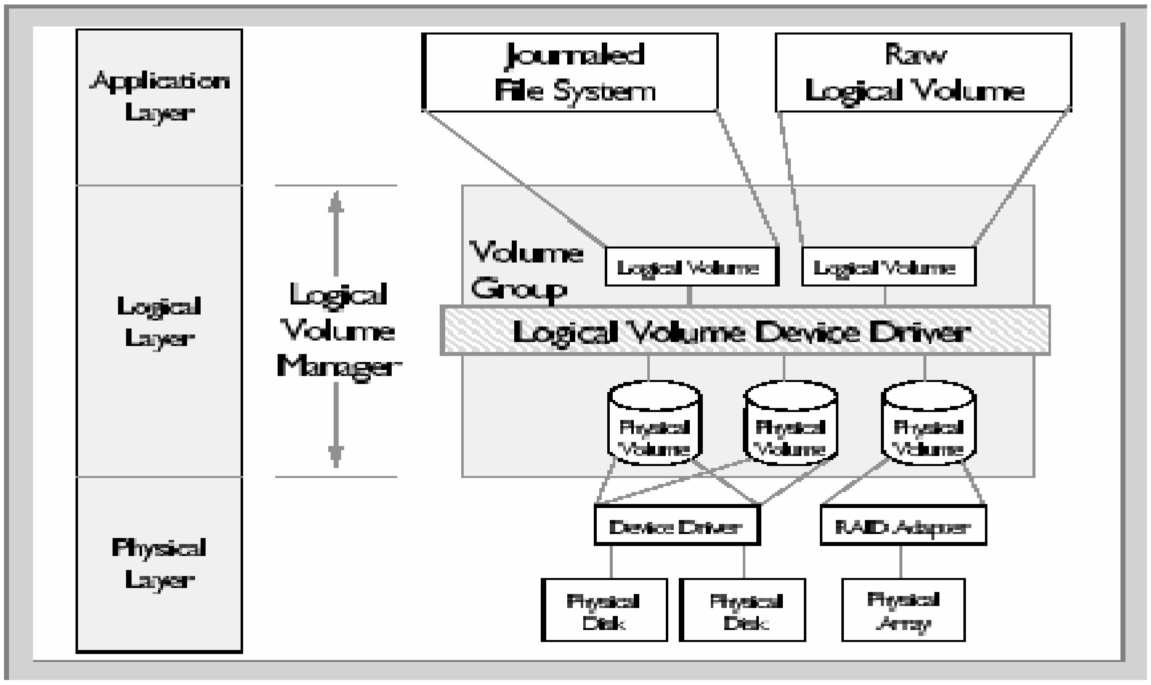


Fig 2.1 How LVM fits between the Physical Layer and the Logical Layer

2.1 Overview:

A hierarchy of structures is used to manage fixed disk storage, and there is a clearly defined relationship (mapping) between them. **Figure 2.2, which shows the relationship between Volume Groups (datavg), Logical Volumes (lv04 and mirrlv), Logical Partitions (LP1,...), Physical Volumes (hdisk9), and Physical Partitions (PP8,...)).** Each individual disk drive is called a physical volume (PV) and has a name, usually /dev/hdiskx (where x is a unique integer on the system). Every physical volume in use belongs to a volume group (VG) unless it is being used as a raw storage device or a readily available spare[18]

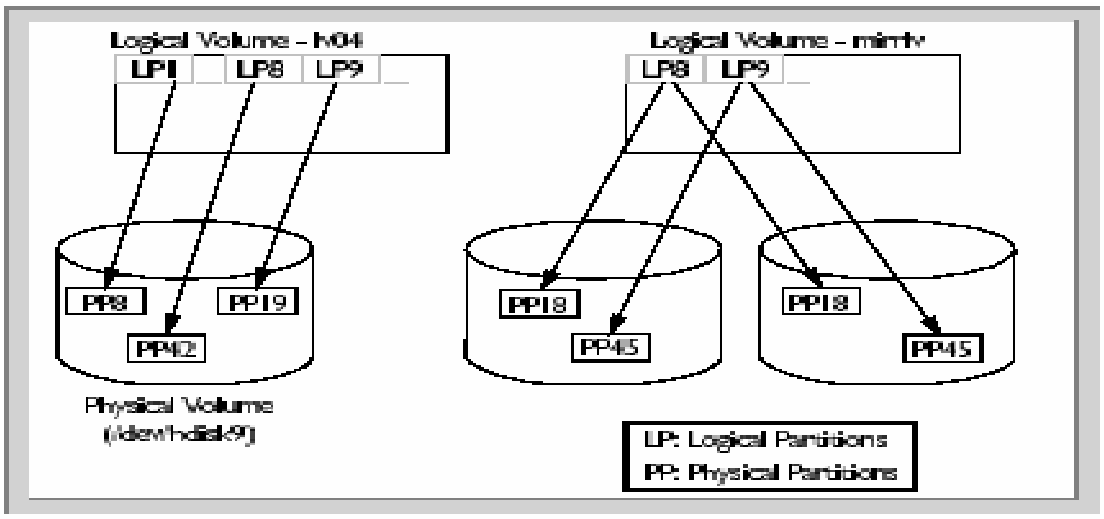


Fig. 2.2 The Relationship between the various components of the LVM

(often called a 'Hot Spare'). Each physical volume consists of a number of disks (or platters) stacked one above the other. Each is divided into physical partitions (PPs) of a fixed size for that physical volume. For space allocation purposes, each physical volume is divided into five regions (outer_edge, outer_middle, center, inner_middle, and inner_edge), and these can be viewed as cylindrical segments cut perpendicularly through the disk platters (**See Figure 2.3**).

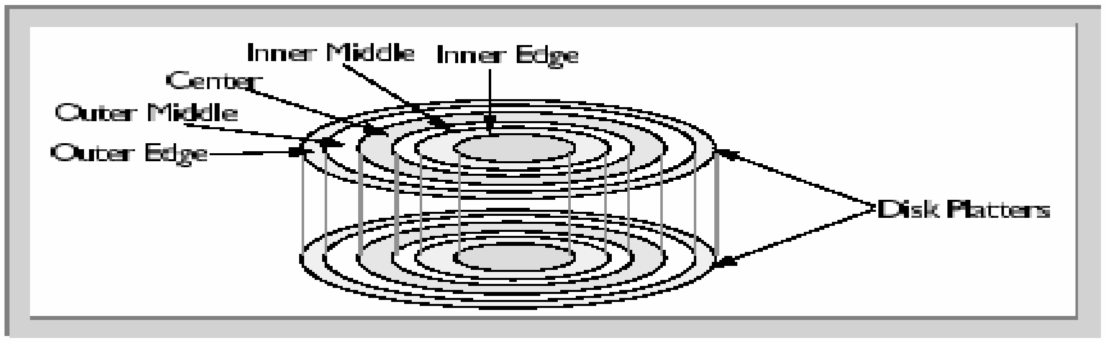


Fig. 2.3 Physical Volume Regions

The number of physical partitions in each region varies depending on the total capacity of the disk drive. The set of operating system commands, library subroutines, and other tools that allow the user to establish and control logical volume storage is called the Logical Volume Manager. The Logical Volume Manager controls disk resources by mapping data between a simple and flexible logical view of storage space and the actual physical disks. The Logical Volume Manager does this by using a layer of device driver code that runs above the traditional physical device drivers. This logical view of the disk storage is provided to applications and is independent of the underlying physical disk structure. After installation, the system has one volume group (the root volume group called rootvg) consisting of a base set of logical volumes required to start the system plus any other logical volumes that were specified to the installation script. Any other physical volumes that are connected to the system can be added to this volume group (using the extendvg command) or used to create a new volume group (using the mkvg command).

The following relationship exists between volume groups and physical volumes:

• On a single system, one to many physical volumes can make up a volume group.

• Physical volumes cannot be shared between volume groups.

• The entire physical volume becomes part of the volume group.

• The LVM is physical volume independent, thus, different types of physical volumes can make up a volume group within the limitation imposed by the partition size and partition limit.

Within each volume group, one or more logical volumes (LVs) are defined. Logical volumes are the way to group information located on one or more physical volumes. Logical volumes are an area of disk used to store data, which appears to be contiguous to the application, but can be non-contiguous on the actual physical volume. It is this definition of a logical volume that allows them to be extended, relocated, span multiple physical volumes, and have their contents replicated for greater flexibility and availability.

Each logical volume consists of one or more logical partitions (LPs). Each logical partition corresponds to at least one physical partition. If the logical volume is mirrored, then additional physical partitions are allocated to store the additional copies of each logical partition. These copies usually reside on different physical volumes (for availability) but, for performance reasons, may reside on the same physical volume.

Till now we have seen the following terms with respect to the logical volume manager: **Figure 2.4 explains the interrelationship between these terms.**

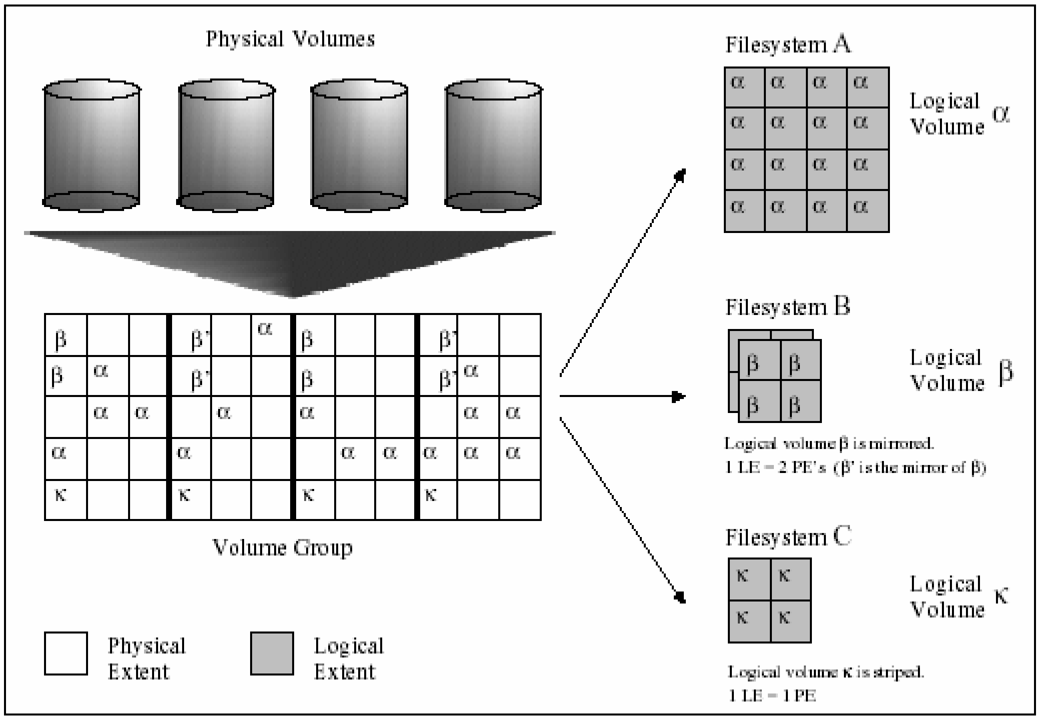


Fig. 2.4 Linux LVM Model

Physical Volume (PV):

A physical volume is any regular hard disk[12]. It can also be a logical drive provided by a hardware RAID controller, which from the operating systems point of view is just a harddisk. Both IDE and SCSI devices are possible. You can even use software RAID to simulate a hard drive, but we recommend a hardware solution is RAID is to be used. A PV can be divided into up to a maximum of 65535 PEs.

Volume Group (VG):

A volume group contains a number of physical volumes (PVs). It groups them into one logical drive – the volume group. The volume group is the equivalent of a hard drive.

Logical Volume (LV):

A logical volume is a part of a volume group [10]. It is the equivalent of a partition. Logical volumes is what really contains the data. Logical volumes are created in volume groups, can be resized within the boundaries of their volume group and even moved to other groups. They are much more flexible than partitions since they do not rely on physical boundaries as partitions on disks do.

Physical Extent (PE):

Logical volumes are made up of physical extents, the smallest physical storage unit in an LVM system. Data is stored in PEs that are part of LVs that are in VGs that consist of PVs. It means any PV consists of lots of PEs, numbered 1-m, where m is the size of the PV divided by the size of one PE. See the command pvdata.

Logical Extent (LE):

Each logical volume is split into chunks of data, known as logical extents. The extent size is the same for all logical volumes in the volume group.

2.2 Physical Volume:

The lowest level in the LinuxLVM 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 backup.

pvscan	*Scan all available SCSI, (E)IDE disks and multiple devices (softwareRAID) for physical volumes.*
pvcreate	*Create a physical volume. Target can be a partition (partition id must be 8e), a whole disk*
pvchange	*Change attributes on physical volumes.*
pvdisplay	*Allows viewing of the attributes of physical volumes.*

pvdata	*Shows debugging information about a physical volume. Uses the (volume group descriptor array (VGDA) of a physical volume.*
pvmove	*Allows moving the logical/physical extents allocated on one logical/physical volume to one or more other physical volumes.*

Table 2.1 List of Physical Volume Commands[2]

Please refer to Appendix B for the Physical Volume Structures.

Physical volumes, also known as direct access storage devices (DASDs), are fixed or removable storage devices. Typically, these devices are hard disks. A fixed storage device is any storage device defined during system configuration to be an integral part of the system DASD. The operating system detects an error if a fixed storage device is not available at some time during normal operation. A removable storage device is any storage device defined during system configuration to be an optional part of the system DASD. The removable storage device can be removed from the system at any time during normal operation. As long as the device is logically unmounted first, the operating system does not detect an error.

2.3 Volume Group:

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. Two of the most important components of the volume group are the volume group descriptor area (VGDA) and volume group status area (VGSA).

The VGDA is an area at the front of each disk, which contains information about the volume group, the logical volumes that reside on the volume group. This VGDA, the metadata is replicated on each disk in the volume group. This VGDA area is also used in quorum voting. The VGDA contains information about what other disks make up the volume group. This information is what allows the user to just specify one of the disks in the volume group when they are using the “importvg” command to import a volume group into an AIX [18] system. The importvg will go to that disk, read the VGDA and find out what other disks (by PVID) make up the volume group and automatically import those disks into the system (and its) ODM as well. The information about neighboring disks can sometimes be useful in data recovery. For the logical volumes that exist on that disk, the VGDA gives information about that logical volume so anytime some change is done to the status of the logical volume (creation, extension, or deletion), then the VGDA on that disk and the others in the volume group must be updated. The following excerpt has been taken from the actual LVM code in Linux:

```
/*
 * VGDA: default disk spaces and offsets
 *
 *   there's space after the structures for later extensions.
 *
 *   offset              what                              size
 *   ---------------     --------------------------------  ------------
 *   0                   physical volume structure         ~500 byte
 *
 *   1K                  volume group structure            ~200 byte
 *
 *   6K                  namelist of physical volumes      128 byte each
 *
 *   6k + n * ~300       n logical volume structures       ~300 byte each
 *   byte
 *   + m * 4byte         m physical extent alloc. structs  4 byte each
 *
 *   End of disk -       first physical extent             typically 4 megabyte
 *   PE total *
 *   PE size
 *
 */
```

The Volume Group Status Area (VGSA) [7]is comprised of 127 bytes, where each bit in the bytes represents up to 1016 Physical Partitions that reside on each disk. The bits of the VGSA are used as a quick bit-mask to determine which Physical Partitions, if any, have become stale. This is only important in the case of mirroring where there exists more than one copy of the Physical Partition. Stale partitions are flagged by the VGSA. Unlike the VGDA, the VGSA's are specific only to the drives, which they exist. They do not contain information about the status of partitions on other drives in the same volume group. The VGSA is also the bit masked used to determine which physical partitions must undergo data resyncing when mirror copy resolution is performed. Please refer to Appendix B for the Volume Group Structures.

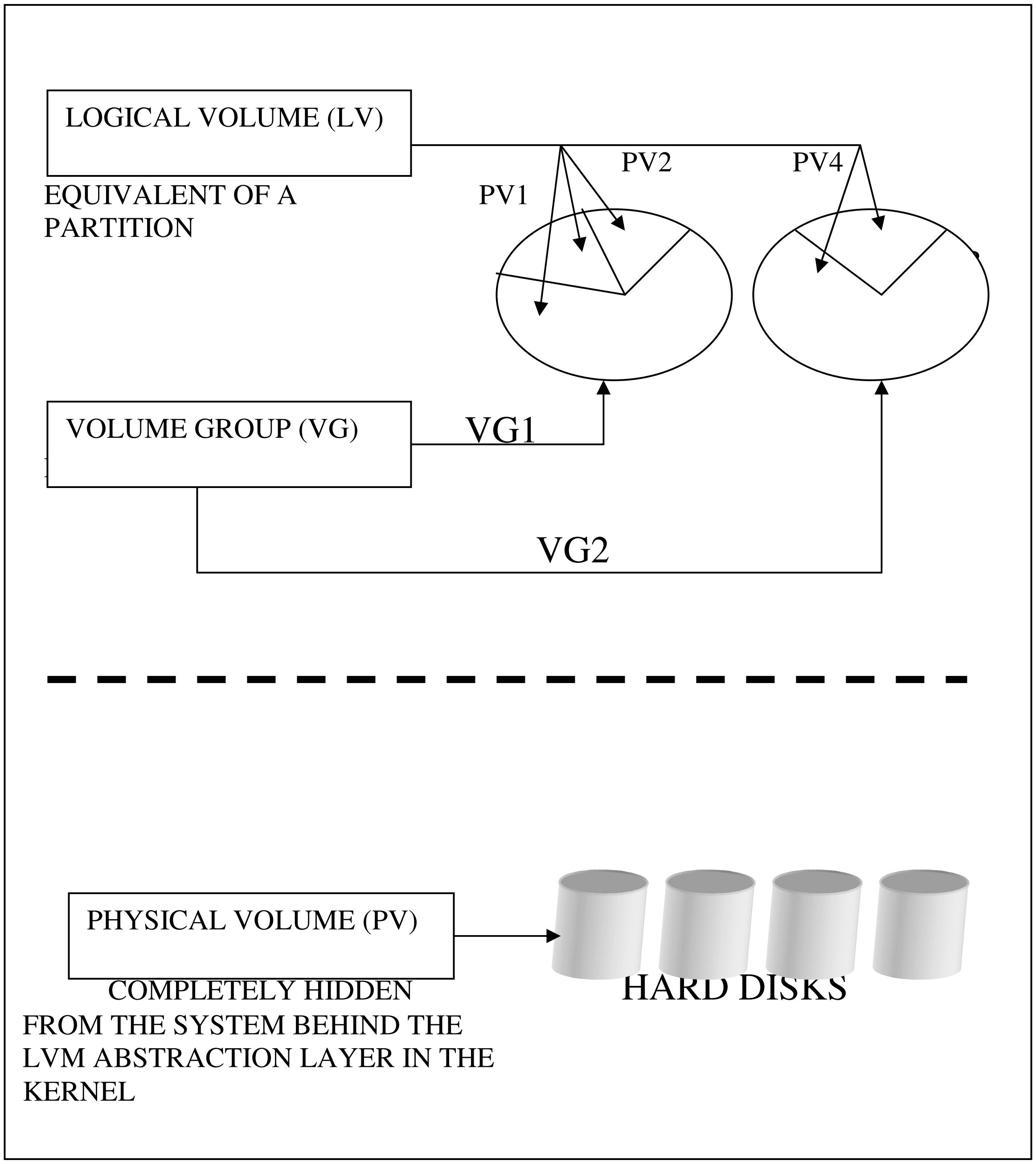

LOGICAL VOLUME (LV)
EQUIVALENT OF A PARTITION
PV1
PV2
PV4
VOLUME GROUP (VG)
VG1
VG2
PHYSICAL VOLUME (PV)
COMPLETELY HIDDEN FROM THE SYSTEM BEHIND THE LVM ABSTRACTION LAYER IN THE KERNEL
HARD DISKS

Fig. 2.5 The Physical Volume, Volume Group and the Logical Volume

A Volume Group (VG) splits each of its underlying Physical Volumes (PV's) into smaller units of Physical Extents (PE's). One PE is usually a few megabytes. 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[2]. A LE is simply a PE, which has been allocated to a Logical Volume. This is illustrated in **Figure 2.5**

vgscan	Scan all disks for volume groups and build /etc/lvmtab and /etc/lvmtab.d/*which are the database for all other lvm commands.
vgdisplay	Display the attributes of a volume group with it's physical and logicalvolumes and their sizes etc.
vgextend	Add physical volumes to a volume group to extent its size.
vgmerge	Merge two volume groups
vgreduce	Remove one or more unused physical volumes from a volume group.
vgrename	Rename a volume group.
vgsplit	Splits physical volume(s) from an existing volume group into a new volume group.
vgremove	Allows you to remove one or more volume groups. The volume group(s) must not have any logical volumes allocated and must also be inactive.
vgchange	Change the attributes of a volume group, e.g. activate it.
vgcreate	Creates a new volume group with at least one physical volume in it.

Table 2.2 List of Volume Group Commands

2.4 Logical Volume:

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 can be created. Creating an LV is done by: lvcreate -L1500 –n test 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 from test vg will be in the /dev/test vg/ directory. Other options can be used to specify an LV, which is striped. Logical volumes are not allowed to be mirrored to other volume groups and cannot be spanned from one volume group to another volume group. This is in part to the information held on a logical volume by the VGDA. The VGDA can't be expected to not only track a logical volume in its own volume group and then additionally track the logical volume (and more important its status) on another volume group. The logical volume control block (lvcb) consists of the first 512 bytes of a logical volume. This area holds important information such as the creation date of the logical volume, information about mirrored copies, and possible mount points in a journaled filesystem. Please refer to Appendix B for the Logical Volume Structures.

lvscan	Scans all known volume groups or all SCSI, (E)IDE disks and multiple devices in the system for defined logical volumes.
lvvcreate	Creates a new logical volume in a volume group.
lvremove	Allows removal of one or more inactive logical volumes.
lvextend	Allows extending the size of a logical volume by allocating unused space in the volume group the logical volume is located in.
lvrename	Renames an existing logical volume.
lvchange	Change the attributes of a logical volume, e.g. activate it.
lvreduce	Allows reducing the size of a logical volume. Be careful when reducing a logical volume's size, because data in the reduced part is lost!
lvdispaly	Allows viewing the attributes of a logical volume like size, read/write

Table 2.3 List of Logical Volume Commands

2.5 Limits:

Max. number of VGs: 99

Max. number of LVs: 256

Max. size of an LV: 512 MB - 2 TB (32 bit), 1 Petabyte (64 bit) depends on PE size

Max. size of storage under LVM control: 256 Terabyte (using all 128 SCSI subsystems)

Max. number of Logical Extents per LV: 65534

Max. number of Physical Extents per PV: 65534

3 Need for a Cluster Logical Volume Manager

Business use of computer systems has increased dramatically over the last half century. The use of the computer has also evolved. In today's business environment, more and more customers are becoming critically dependent on their information technology resources. As a result, they demand that these resources are always available. Any outage has serious business implications including lost revenue and lost business[15]. At the extreme, an extended system outage can cause a business to be permanently closed. The cost of one hour of system downtime can range from tens of thousands of dollars to several million dollars, depending on the nature of the business. Therefore, users require that their system services be continuously available, that is that the services be available 24 hours a day, 365 days a year. Technology that supports increased computer system availability has become critical to many businesses [20].

The key technology that enables us to provide continuous availability is clustering. A *cluster* [14] is a collection of one or more complete systems that work together to provide a single, unified computing capability. The perspective from the end user is that the cluster operates as though it were a single system. Work can be distributed across multiple systems in the cluster. Any single outage (planned or unplanned) in the cluster will not disrupt the services provided to the end user. End user services can be relocated from system to system within the cluster in a relatively transparent fashion.

The logical volume manager available currently is not cluster-aware and if used in a cluster then it is very much likely that we might end up losing all the data. In a fibre-channel or shared-SCSI environment where more than one machine has physical access to a set of disks, one can use LVM to divide these disks up into logical volumes. Sharing of data is simpler and more efficient in a cluster file system like GFS. The key thing to remember when sharing volumes is that all the LVM administration must be done on one node only and that all other nodes must have LVM shut down before changing anything on the admin node. Then, when the changes have been made, it is necessary to run vgscan on the other nodes before reloading the volume groups. Also, unless the system is running a cluster-aware filesystem (such as GFS) or application on the volume, only one node can mount each filesystem. It is up to system administrator to enforce this explicitly, LVM will not stop you corrupting your data.

If you need to do any changes to the LVM metadata (regardless of whether it affects volumes mounted on other nodes) you must go through the following sequence. In the steps below ‘‘admin node” is any arbitrarily chosen node in the cluster [10].

```
Admin node          Other nodes
----------          -----------
                    Close all Logical volumes (umount)
                    vgchange -an
<make changes
eg lvextend>
                    vgscan
                    vgchange -ay
```

As the current LVM does not support distributed updates and locking on metadata so you must be extra careful when making changes to logical volumes. If you need to make any changes to logical volumes, you must do it with only one host online. In the current scenario you can successfully do LVM metadata updates in a cluster without rebooting nodes by unmounting all LVM partitions on all nodes but one and also removing the SCSI module (just to make sure nothing is cached). After performing the LVM modifications on the remaining node, It is then required to re-probe the SCSI module on the other nodes and rerun vgscan after which the new volumes are detected.

This seems to be a very complex procedure and what appears to be needed for better cluster support is at least some simple locking on the metadata to enforce the single node exclusive access to cluster metadata when performing changes, because at the moment there are no checks on operations which could potentially corrupt data. The traditional volume managers on HP-UX, Solaris (VxVM) and AIX [18][19] do not usually support shared access to a volume group from two or more nodes in a cluster even if the nodes access different logical volumes. This is done explicitly to prevent the kind of problems like the chance that two nodes have different in-core metadata about the

VG. HP's LVM supports a read-only vgchange that allows only read-only access to the VG and its LV's. In these traditional environment, the clustering software should export and import the VG's as necessary, and run some clusterwide resource manager that takes care of who currently "owns" the VG and a distributed lock manager to maintain consistency. Veritas has a special Cluster Volume Manager (CVM) that allows shared access to volume groups, but it is only used with parallel databases such as Oracle Parallel Server.

At present one would put a procedure in place such that any LVM changes done from a particular node requires the bouncing of VGs on all other attached nodes. Fortunately, after initial cluster setup, manipulation of LVs won't really be performed on a routine basis. It's entirely possible to run a non-shared-media-aware filesystem as long as no more than one cluster node has a given file system mounted at a time.
To illustrate:

```
|-------- VG --------|
||====== LV0 =======||
|| (ext2)           || --> Mounted on Cluster Node 1
||==================||
||====== LV1 =======||
|| (ext2)           || --> Mounted on Cluster Node 2
||==================||
||====== LV2 =======||
|| (ext2)           || --> Mounted on Cluster Node 3
||==================||
||====== LV3 =======||
|| (ext2)           || --> Mounted on Cluster Node 4
||==================||
|                    |
|  Free Space in VG  |
|                    |
|====================|
```

Because none of the cluster nodes are attempting to share access to the actual blocks where each filesystem is stored, there are no concurrency issues. One can use the benefits of LVM to unmount LV0's fs on Cluster Node 1, resize the LV, resize the fs and remount. Now, Cluster Node's 2, 3 and 4 need to have their in-core LVM metadata updated in order to see the new size of LV0.Once this is done via the vgchange bounce, everything is consistent.

4 What is a Cluster Logical Volume Manager?

In a cluster environment, there are many independent systems and each of these systems will have their own logical volume managers and a cluster file system like Global File System (GFS). The LVMs of the different nodes in the cluster will access the disks from a pool of shared disks. The need for a cluster logical volume manager (CLVM) arises because the different LVMs may access disks in volume groups concurrently which may result in an inconsistent state. In order to avoid this, we need to run a distributed lock manager (DLM) over various nodes to maintain consistency. The LVMs of the different nodes in the cluster along with the DLM constitute a Cluster Logical Volume Manager.

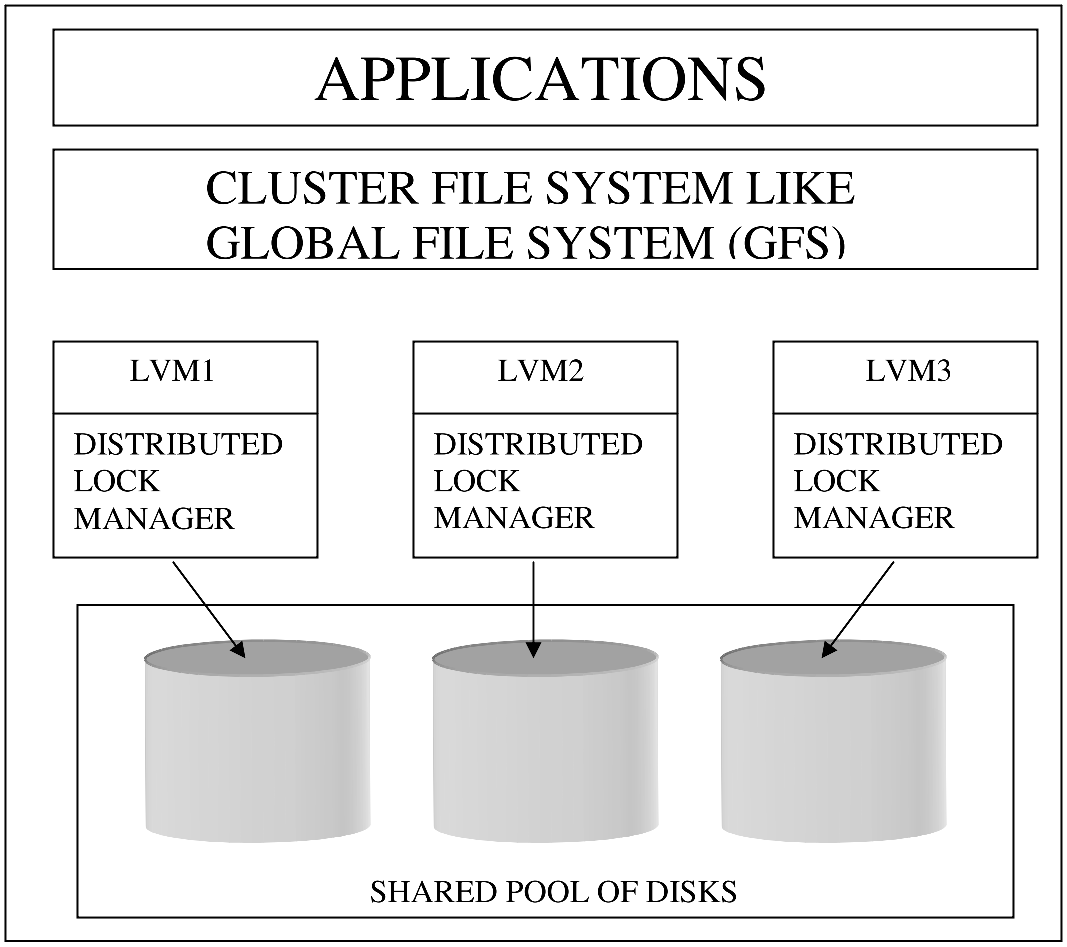


Fig. 4.1 Components of a Cluster Logical Volume Manager

Any volume management system consists of disk operations involving data and metadata. In a clustered environment, all the data specific operations are taken care of by the cluster file system. The job of the cluster logical volume manager is to manage all the meta data on the disks. This meta data in the form of volume group descriptor area (VGDA) resides on all the disks in the volume group. When any node decides to change the structure of any volume group then it is the duty of the CLVM to instruct the distributed lock manager to lock all the disks belonging to the volume group. All the data operations on these disks are queued up while all the operations on the other volume groups can continue in the normal fashion. Once the DLM has locked the disks, the CLVM makes the related changes on the disks in the volume group, initializes the meta data and puts the updated copy of the meta data on all the disks. The DLM now unlocks these disks and all the queued operations resume.

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.

5 Design

The Cluster Logical Volume Manager will be designed by modifying the existing logical volume manager, which exists in Linux. The single most important component of this design is the cluster lock manager, which will handle all concurrent accesses to disks.

5.1 Cluster Lock Manager:

The cluster lock manager (CLM) is a daemon subsystem that provides advisory locking services that allow concurrent applications running on multiple nodes in a cluster to coordinate their use of shared resources, mainly the shared volume groups. The cluster lock manager provides a single, unified lock image shared among all nodes in the cluster. Each node runs a copy of the lock manager daemon. These lock manager daemons communicate with each other to maintain a cluster-wide database of locked resources and the locks held on these locked resources Note that all locks are advisory. The system does not enforce locking. Instead, applications running on the cluster must cooperate to work together. An application that wants to use a shared resource is responsible for first obtaining a lock on that resource, then attempting to access it.

To start the cluster lock manager, you have to specify the necessary option during the cluster manager startup.

```
root@abhinay:/ # command for starting CLM
Cluster Services
Start Cluster Services
```

To stop the cluster lock manager, you have to specify the necessary option during the cluster manager shutdown.

```
 root@abhinay:/ # command for stopping CLM
Cluster Services
Stop Cluster Services
```

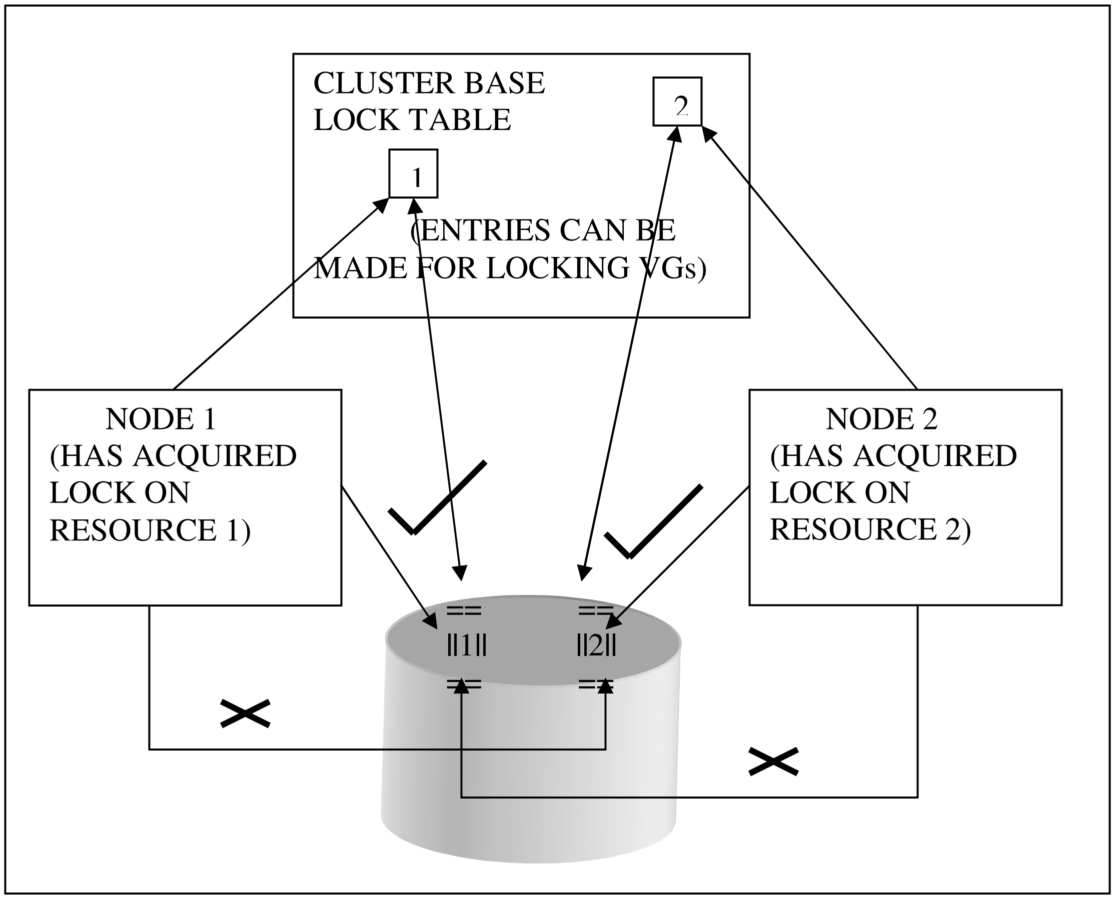
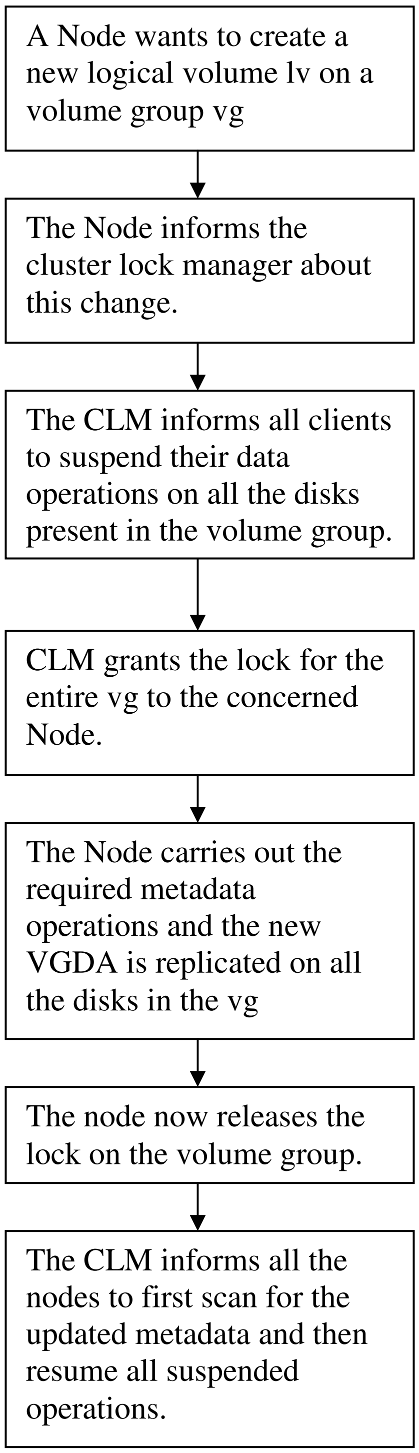

CLUSTER BASE
LOCK TABLE
1
2
(ENTRIES CAN BE MADE FOR LOCKING VGs)
NODE 1 (HAS ACQUIRED LOCK ON RESOURCE 1)
NODE 2 (HAS ACQUIRED LOCK ON RESOURCE 2)
||1||
||2||

Fig. 5.1 Cluster Lock Manager

As shown in **Figure 5.1** each node can acquire a lock on a resource and then that resource cannot be accessed by any other node [18]. Node 1 has locked resource 1 and this entry is made in the cluster base lock table. If Node 2 tries to access this resource then it will try and acquire a lock for it. It requests the CLM for the required mode lock, the CLM scans the lock table and sees that Node 1 has already acquired a lock for resource 1. Hence Node 2 is denied access and put in a queue. (If the lock mode requested by Node 2 is compatible with the lock mode of Node 1 then the CLM allows Node 2 to access the resource 1). One Node 1 has released the lock on resource 1 then the CLM scans through the queue to check whether any node has requested a lock on that particular resource. If it finds that some node had requested for a lock then it grants the lock to the node.

5.2 Sequence Diagram of a CLVM:

Let us now try and understand what exactly happens when metadata updations have to take pace in a concurrent environment.


A Node wants to create a new logical volume lv on a volume group vg
The Node informs the cluster lock manager about this change.
The CLM informs all clients to suspend their data operations on all the disks present in the volume group.
CLM grants the lock for the entire vg to the concerned Node.
The Node carries out the required metadata operations and the new VGDA is replicated on all the disks in the vg
The node now releases the lock on the volume group.
The CLM informs all the nodes to first scan for the updated metadata and then resume all suspended operations.
Fig. 5.2 Sequence Diagram of a CLVM

5.3 Modes of Operations in a CLVM:

Generally speaking, from the logical volume management perspective, there are two categories of data operations on the volume groups.

• LVM configuration operations (Metadata operations)

• I/O operations to the logical volumes (data operations)

The I/O operations to logical volumes are controlled by the cluster file system (GFS). The Cluster Logical Volume Manager comes into the picture only when metadata operations are involved. The two possible modes of operations are:

5.3.1 Idle Mode:

In this mode, the only task of the CLVM is the map the data requested by the cluster filesystem to the physical volume within a volume group. No locking of volume groups is required as metadata operations are not involved. Issues like concurrent accesses to same physical volumes are handled by the file system.

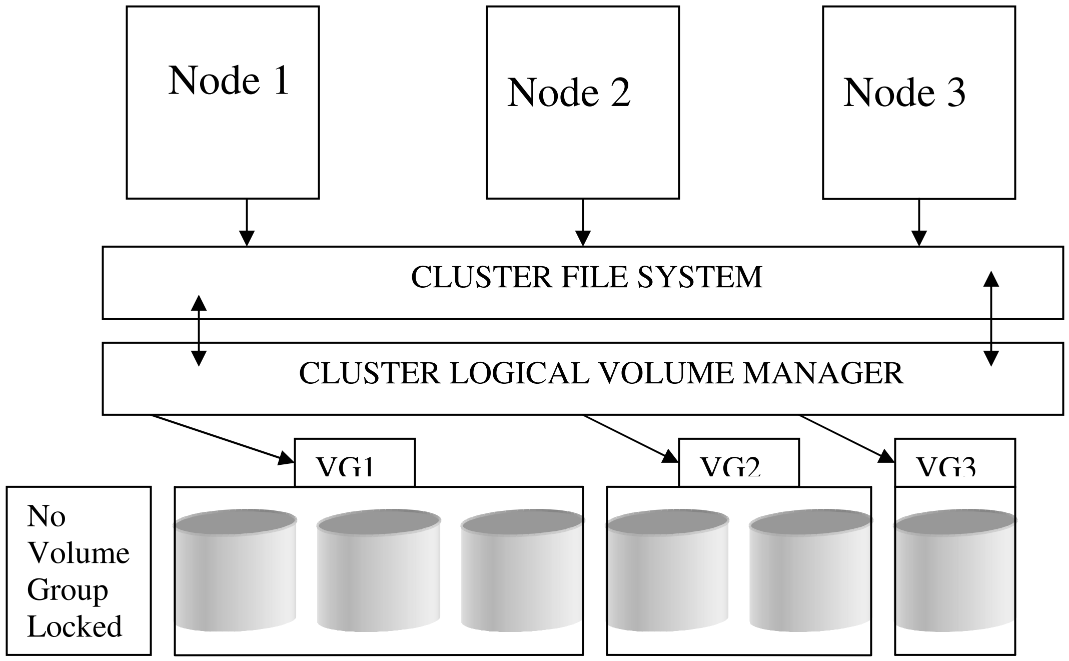


Fig. 5.3 Idle Mode of Operation of CLVM

For example, consider a function like getblk( ) which requests the filesystem to get a block of data from the disk. The CLVM maps the block in the logical volume to a physical block and returns the block back to the file system.

5.3.2 Active Mode:

In this mode, the task of the CLVM is to lock the disks in the volume group, perform the metadata operations and then unlock the disks in the volume group. It is important to note that it is necessary to *lock all the disks in the volume group* because the VGDA i.e. the metadata is replicated on all the disks in the volume group.

For example, consider a command like lvcreate (vg1) which requests the CLVM to create a logical volume on disk1 of volume group vg1. The CLVM then requests the lock manager to lock all the three disks in vg1 and queue all data operations on this particular volume group.

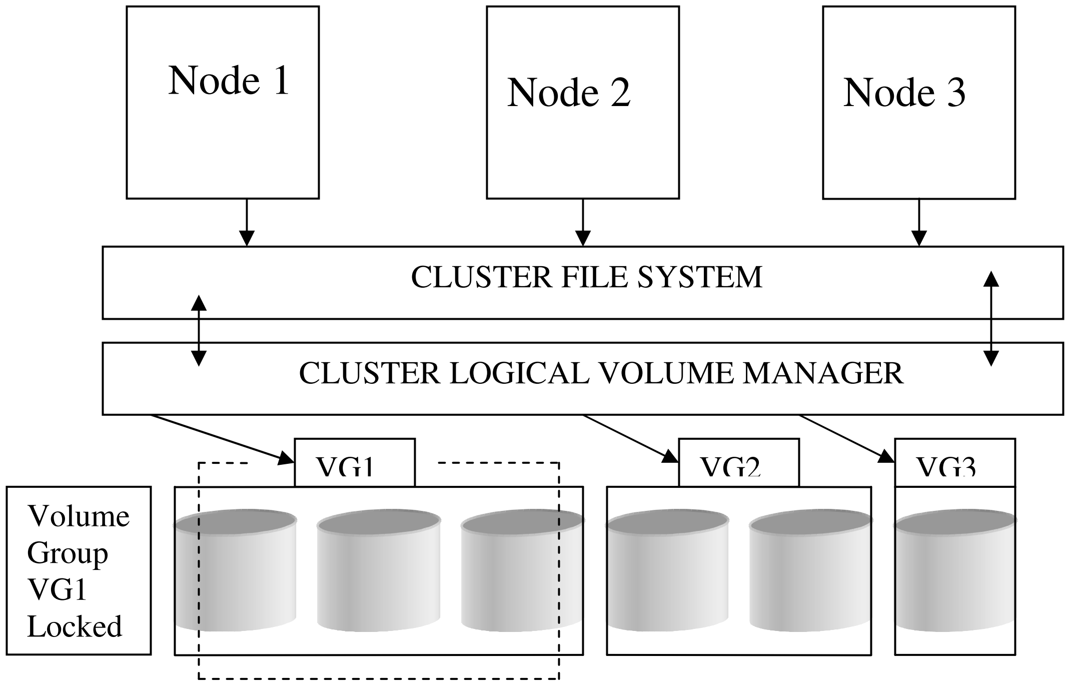


Fig. 5.4 Active Mode of Operation of CLVM

6 Related Work

6.1 LVMs under Linux and other Platforms

There are many logical volume managers under the linux environment [1]. Each of these LVMs is implemented in a slightly different manner but all of them ultimately give some mapping from physical volume to logical volume through various layers of abstraction.

6.1.1 Linux LVM:

Constructing flexible virtual views of storage with LVM is possible because of the well- defined abstraction layers [7]. The lowest level in the Linux 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. Multiple Physical Volumes (initialized partitions) are merged into a Volume Group (VG). This is done with the command: vgcreate 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.

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 else- where. There are also many more commands to rename, remove, split, merge, activate, de- activate and get extended information about current PV's, VG's and LV's.

6.1.2 GFS's Pool Driver:

The Pool driver is a simple Linux Volume Manager, which has been developed in conjunction with the Global File System [12]. The Pool driver knows the physical distribution of Device Memory Export Protocol (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 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

The Pool Driver has the ability to configure a Pool device with distinct "sub-pools" consisting of particular partitions. 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 subplots (e.g. four sets of four striped disks rather than 16 striped disks).

6.1.3 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 [19]. 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. The VxVM RAID-0 (striping) implementation allows a striped plex to be expanded. 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.

6.1.4 Sun Solstice DiskSuite:

Sun's Solstice DiskSuite (SDS) is a volume manager, which runs on the Solaris operating system only [11]. 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.

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, man- age or use shared devices.

6.1.5 Free BSD Vinum:

Vinum is a volume manager implemented under FreeBSD and is Open Source like the Linux Volume Managers [4]. 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 sub- disk 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.

6.1.6 IBM EVMS:

The EVMS architecture is based upon the LVMS architecture developed by IBM [9]. As such, it consists of two basic components: the LVM Engine and the LVM Runtime. The LVM Engine runs in user space, while the LVM Runtime runs in kernel space. The LVM Runtime allows the operating system to access and use properly configured volumes. The creation, configuration, and management of volumes, volume groups, partitions, and the disks they reside on is handled by the LVM Engine. Thus, the LVM Engine handles setup and initialization, while the LVM Runtime handles the actual use of the volumes. This division of labor between the LVM Runtime and the LVM Engine is designed to reduce the size of, and the kernel resources required by, the LVM Runtime. The LVM Engine is intended to be a shared object, meaning that a single instance of the LVM Engine can be used by multiple programs. These services are then used by other programs to manipulate the disks and volumes in the system. The LVM Engine is not intended to be the user interface to the EVMS, but a programmatic interface to the EVMS. It provides a robust and complete interface, and is the only interface to the EVMS. All user interfaces and programs requiring EVMS support must go through the LVM Engine. One of the unique features of the EVMS is its ability to accept plug-in modules. A plug-in module consists of executable code, which can be loaded and run by the EVMS. Plug-in modules allow the capabilities of the EVMS to be expanded without having to alter the code of the EVMS itself. This simplifies development and maintenance while increasing the stability of the EVMS over time. This is also the main mechanism for allowing the EVMS to emulate other LVM systems.

6.2 EnterpriseVolume Management System:

IBM's Enterprise Volume Management System (EVMS) is convenient for solving the problem of migrating from one operating system to another [9]. It has many useful attributes, which make it worthwhile as a general purpose logical volume management system. It provides unparalleled flexibility and extensibility while remaining efficient, and is an extension of an existing logical volume management system which has been implemented, to varying degrees, on existing IBM operating systems.

EVMS consists of two basic components: the LVM Engine and the LVM Runtime. The LVM Engine runs in user space, while the LVM Runtime runs in kernel space. The LVM Runtime allows the operating system to access and use properly configured volumes. The creation, configuration, and management of volumes, volume groups, partitions, and the disks they reside on is handled by the LVM Engine. Thus, the LVM Engine handles setup and initialization, while the LVM Runtime handles the actual use of the volumes. This division of labor between the LVM Runtime and the LVM Engine is designed to reduce the size of, and the kernel resources required by, the LVM Runtime.

The EVMS architecture calls for five classes of plug-in modules, executable code that can be loaded and run by the EVMS: Device Managers, Partition Managers, Volume Group Emulators, Features and Filesystem Interface Modules. Device Managers are used to communicate and manage the storage devices available to the system. Partition Managers are used to recognize and manipulate the partition information on a device. Volume Group Emulators (VGEs) are used when volume group support is desired, or when emulating a volume group based LVM. Features are used to assemble volumes, and provide specific capabilities on the volumes. Finally, Filesystem Interface Modules allow the EVMS to perform basic filesystem tasks, such as formatting, checking, and resizing, as well as gather information about the filesystem in use on a volume before performing operations that would have an effect upon the filesystem (volume expansion or shrinking, for instance). Examples of device managers would be a local storage manager (for accessing local storage devices), and a SAN manager, which would allow the system to access and use storage on a Storage Area Network. Examples of partition managers would be a DOS partition manager (for accessing disks partitioned by DOS, Windows or OS/2), and a Macintosh partition manager (for accessing disks

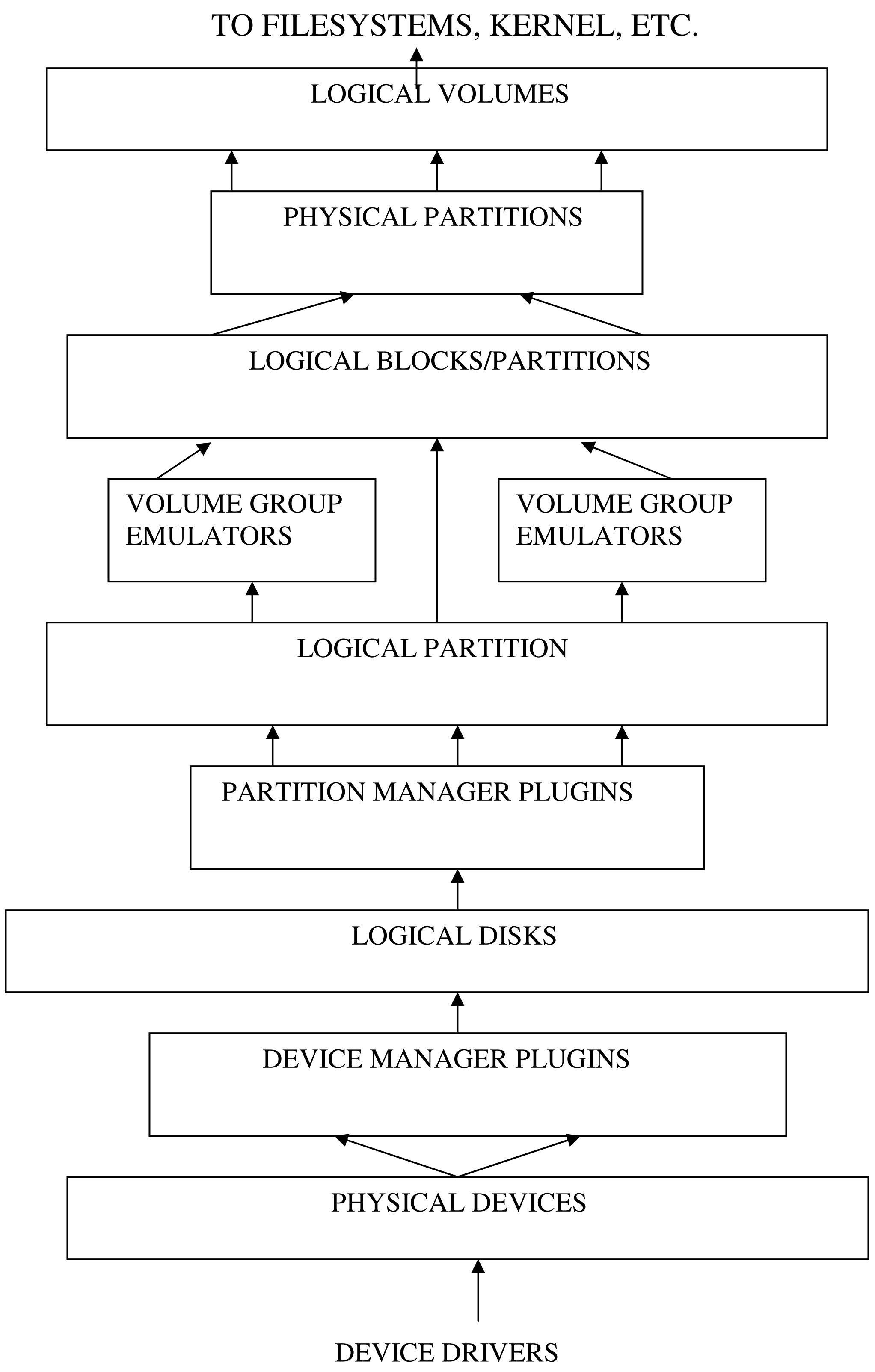


Fig. 6.1 Design of Enterprise Volume Management System (EVMS)

partitioned by an Apple Macintosh). An example of a VGE would be an AIX VGE (for accessing and manipulating volume groups created by the LVM in the AIX operating system). Examples of Features would be drive linking, RAID 0, RAID 5, mirroring, encryption, etc.

The EVMS architecture consists of various layers of abstractions. The first is that of the logical disk. This abstraction was introduced into the EVMS to provide a standard, consistent method for accessing the storage devices available to the system. Anything which can be made to appear as a logical disk can be managed by this EVMS. This can include storage obtained from a Storage Area Network (SAN) or other network attached device, for instance. The next layer of abstraction is the logical partition. This abstraction was created so that the EVMS was not tied to any particular partitioning scheme (many exist). The EVMS uses a generic memory management structure for tracking meta data, partitions and free space on a logical disk. Thus, all logical partitions appear the same regardless of the actual underlying partitioning scheme used. Volume groups are a construct found in many LVM systems. The Volume Group abstraction was added to the EVMS to allow it to emulate volume group based LVMs found on other operating systems. Volume groups are constructed from one or more logical disks and/or logical partitions. The EVMS uses a generic structure for representing volume groups, and for tracking which logical disks and/or logical partitions belong to a volume group. Thus, all volume groups appear the same regardless of the actual underlying representation used.
The last layer of abstraction is the logical volume. The logical volume is what is visible to the operating system. The operating system should only recognize logical volumes and physical devices. Any partitions which may exist on the physical devices should not be directly accessible to the operating system. Thus, all I/O to partitions and volumes must pass through the EVMS.

The EVMS architecture can be successfully implemented on various operating systems. This architecture offers many advantages over previous designs, particularly in the areas of expandability, flexibility and maintenance, and represents a new approach to logical volume management.

7 Conclusion

The finer details of a logical volume manager and its shortcomings when used in a cluster have been clearly understood. The proposed design shows how the existing LVMs can be leveraged to become cluster aware. The role of a cluster lock manager in a CLVM is very important. The cluster lock manager provides a single, unified lock image shared among all nodes in the cluster. Each node runs a copy of the lock manger daemon.

The two modes of the CLVM were discussed. It can be observed that the function of the CLVM in the 'idle mode' is only to provide mapping of logical to physical volume, whereas in the 'active mode' where metadata operations are involved the CLVM ensures that the cluster lock manager locks the requisite volume group.

The need of a Cluster Logical Volume Manager is paramount today. Using the logical volume manager in a cluster environment is a very complex task and involves many steps. Each of these must be carried out carefully; else it might result in the entire data getting corrupt. I have proposed a design for a Cluster Logical Volume Manager for the Linux LVM that can also be used with different logical volume managers under various platforms.

Currently there is ongoing research at various organizations for developing a CLVM. Veritas's Cluster Volume Manager and IBM's Enterprise Volume Management System are under development. The logical volume manger in Linux is also being enhanced with cluster features. Some of these features have been incorporated in logical volume manager 2 (LVM2) for Linux.

Appendix A

Code Modification in LVM:

The following code is from the LVM source code lvm_1.0.4.tar.gz [5] in Linux present in file vgcreate.c and an executable of this file is run in response to vgcreate system call.

```
/* create VGDA in kernel */
  if ( opt_v > 0) printf ( "%s -- creating VGDA for volume group \"%s\" "
                          "in kernel\n",
                          cmd, vg_name);
  if ( ( ret = vg_create ( vg_name, &vg)) < 0) {
     fprintf ( stderr, "%s -- ERROR \"%s\" creating VGDA for volume "
                      "group \"%s\" in kernel\n",
                      cmd, lvm_error ( ret), vg_name);
     if ( ret == -LVM_EVG_CREATE_REMOVE_OPEN)
        fprintf ( stderr, "%s -- LVM not in kernel?\n", cmd);
     fprintf ( stderr, "\n");
     return LVM_EVGCREATE_VG_CREATE;
  }
  if ( opt_v > 0) printf ( "%s -- inserting volume group \"%s\" into \"%s\"\n",
                          cmd, vg_name, LVMTAB);
  if ( ( ret = lvm_tab_vg_insert ( vg_name)) < 0) {
     fprintf ( stderr, "%s -- ERROR \"%s\" inserting volume group \"%s\" "
                      "into \"%s\"\n\n",
                      cmd, lvm_error ( ret), vg_name, LVMTAB);
     return LVM_EVGCREATE_VG_INSERT;
  }         ◄────────────

  lvm_interrupt ();
  LVM_UNLOCK ( 0);
  printf ( "%s -- volume group \"%s\" successfully created and activated\n\n",
          cmd, vg_name);
  return 0;
}
```

The code initially performs a whole lot of error checks to confirm whether all the supplied information is right and the volume group can be created. If it is indeed possible to correctly create the volume group then it calls the function lvm_interrupt( ) which calls the kernel mode functions to create the volume group. After the volume group has been created the function LVM_UNLOCK(0) is called to indicate that all regular operations can now proceed. Inorder to modify this code so that it becomes cluster aware is that another function like Cluster_lock_manager_invoke( ) should be called to lock the volume group before the interrupt( ) function is called,. This function should be placed at the position of the $\longleftarrow$

Appendix B

LVM Structures:

The following section gives the actual structures used in the Logical Volume Manager implementation in Linux for the Physical Volume, Volume Group and the Logical Volume [5].

Physical Volume Structures:

```
typedef struct pv_disk {
        char id[2];                                 /* Identifier */
        unsigned short version;                     /* HM lvm version */
        lvm_disk_data_t pv_on_disk;
        lvm_disk_data_t vg_on_disk;
        lvm_disk_data_t pv_uuidlist_on_disk;
        lvm_disk_data_t lv_on_disk;
        lvm_disk_data_t pe_on_disk;
        char pv_name[NAME_LEN];
        char vg_name[NAME_LEN];
        char system_id[NAME_LEN];                   /* for vgexport/vgimport */
        kdev_t pv_dev;
        uint pv_number;
        uint pv_status;
        uint pv_allocatable;
        uint pv_size;                               /* HM */
        uint lv_cur;
        uint pe_size;
        uint pe_total;
        uint pe_allocated;
        uint pe_stale;                              /* for future use */
        pe_disk_t *pe;                              /* HM */
        struct block_device *bd;
        char pv_uuid[UUID_LEN+1];
        #ifndef __KERNEL__
        uint32_t pe_start;                          /* in sectors */
        #endif
} pv_disk_t;
```

Volume Group Structure:

```
typedef struct vg_disk {
        uint8_t vg_uuid[UUID_LEN];                 /* volume group UUID */
        uint8_t vg_name_dummy[NAME_LEN-UUID_LEN];
        uint32_t vg_number;                        /* volume group number */
        uint32_t vg_access;                        /* read/write */
        uint32_t vg_status;                        /* active or not */
        uint32_t lv_max;                           /* maximum logical volumes */
        uint32_t lv_cur;                           /* current logical volumes */
        uint32_t lv_open;                          /* open    logical volumes */
        uint32_t pv_max;                           /* maximum physical volumes */
        uint32_t pv_cur;                           /* current physical volumes FU */
        uint32_t pv_act;                           /* active physical volumes */
        uint32_t dummy;
        uint32_t vgda;                             /* volume group descriptor arrays FU */
        uint32_t pe_size;                          /* physical extent size in sectors */
        uint32_t pe_total;                         /* total of physical extents */
        uint32_t pe_allocated;                     /* allocated physical extents */
        uint32_t pvg_total;                        /* physical volume groups FU */
} vg_disk_t;
```

Logical Volume Structures:

```
typedef struct lv_disk {
        uint8_t lv_name[NAME_LEN];
        uint8_t vg_name[NAME_LEN];
        uint32_t lv_access;
        uint32_t lv_status;
        uint32_t lv_open;                               /* HM */
        uint32_t lv_dev;                                /* HM */
        uint32_t lv_number;                             /* HM */
        uint32_t lv_mirror_copies;                      /* for future use */
        uint32_t lv_recovery;                           /*       "        */
        uint32_t lv_schedule;                           /*       "        */
        uint32_t lv_size;
        uint32_t lv_snapshot_minor;                     /* minor number of original */
        uint16_t lv_chunk_size;                         /* chunk size of snapshot */
        uint16_t dummy;
        uint32_t lv_allocated_le;
        uint32_t lv_stripes;
        uint32_t lv_stripesize;
        uint32_t lv_badblock;                           /* for future use */
        uint32_t lv_allocation;
        uint32_t lv_io_timeout;                         /* for future use */
        uint32_t lv_read_ahead;                              /* HM */
} lv_disk_t;
```

Appendix C

LVM Commands:

This section gives an overview of some of the high level and intermediate level commands of a logical volume manager [7].

High level commands:

cfgvg, chlv, chpv, chvg, cplv, exportvg, extendlv, extendvg, importvg, lslv*, lsvg*, lsvgfs*, migratepv, mklv, mkvg, reducevg, replacepvg, reorgvg, rmlv, rmlvcopy, syncvg, synclvodm, updatelv, updatevg, varyoffvg, varyonvg*

* indicates that the command is a binary

Intermediate level commands:

getlvcb, getlvname, getlvodm, getvgname, lvgenmajor, lvgenminor, lvrelmajor, lvrelminor, putlvcb, putlvodm, lchangelv, lcreatelv, ldeletelv, lextendlv, lquerylv, lreducelv, lresynclv, lchangepv, ldeletepv, linstallpv, lquerypv, lresyncpv, lcreatevg, lqueryvg, lqueryvgs, lvaryonvg, lvaryoffvg, lresynclp, lmigratelv, lmigratepp.

References

[1] ‘Volume Managers in Linux’, David Teigland, Heinz Mauelshagen, Sistina Software, Inc., Presented at 2001 FREENIX Track Technical Program.

[2] ‘The Logical Volume Manager (LVM)’, SuSe White paper, Heinz Mauelshagen, http://www.sistina.com/lvm/.

[3] ’Maintaining Shared LVM Components’, http://www.tu-darmstadt.de/hrz/unix/ibm/documentation/HACMP/admin/ch04.html#200401

[4] ‘The Vinum Volume Manager’, Greg Lehey, January 2000 http://www.vinumvm.org

[5] LVM source code, lvm_1.0.4.tar.gz, http://www.slackware.at/data/slackware-8.1/source/ap/lvm/

[6] LVM2 source code, lvm2_0.95.05.orig.tar.gz, http://packages.debian.org/unstable/admin/lvm2.html

[7] ‘Logical Volume Manager Basics’ www.share.org/proceedings/sh92mo/data/S5338.PDF

[8] ‘A Case for Redundant Arrays of Inexpensive Disks (RAID)’ Katz, Proceedings of the 1988 ACM SIGMOD Conference of Management of Data, June 1988

[9] ‘Emulating Multiple Logical Volume Management Systems within a Single Volume Management Architecture’, Cuong Tran http://oss.software.ibm.com/developerworks/opensource/evms/doc/EVMS White Paper 1.htm

[10] ‘LVM HOWTO’, Sistina http://www.sistina.com/ lvm/doc/ lvm howto/index.html

[11] Solstice DiskSuite 4.0, SUN, Administration Guide, http://docs.sun.com

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

[13] ‘UNIX Internals’, Vahalia Uresh Vahalia, O’reilly Publications, 1996.

[14] ‘In Search of Clusters’, Pfister, Gregory F., Prentice-Hall, Inc., 1998.

[15] ‘Clustering’, Mike Snyder, IBM White Paper for iSeries and AS/400.

[16] ‘IBM DB2 Universal Database clustering, High availability with WebSphere Edge Server’, Maheshwar Rao, WebSphere Consultant, iS3C Consultancy Services Ltd, May 2002.
[17] ‘How to Apply Shared Network Storage to Improve the Speed and Efficiency of Linux Compute Clusters’, Matthew T. O'Keefe, Ph.D.Founder and Chief Technical Officer of Sistina Software, June 28, 2002.
[18] ‘AIX Logical Volume Manager,from A to Z:Introduction and Concepts’, Laurent Vanel, Ronald van der Knaap, Dugald Foreman, Keigo Matsubara, Anton,January2000. www.redbooks.ibm.com
[19] ‘Quick Reference: AIX Logical Volume Manager and Veritas Volume Manager’, IBM Redpaper, October 2000.
[20] ‘Linux Clustering and Storage Management’, Peter J. Braam, CMU, Stelias Computing, Red Hat.
[21] ‘The Design of the Unix Operating System’, Bach, Maurice., Prentice Hall, 1987.

[22] ‘Linux Device Drivers’, Rubini, Prentice Hall, 2001

[23] ‘Stable Storage Architecture’
ftp://ftp.t10.org/t10/drafts/s3p/s3p-r05b.pdf

[24] ‘Small Computer System Interface’
ftp://ftp.t10.org/t10/drafts/sbc2/sbc2r06.pdf