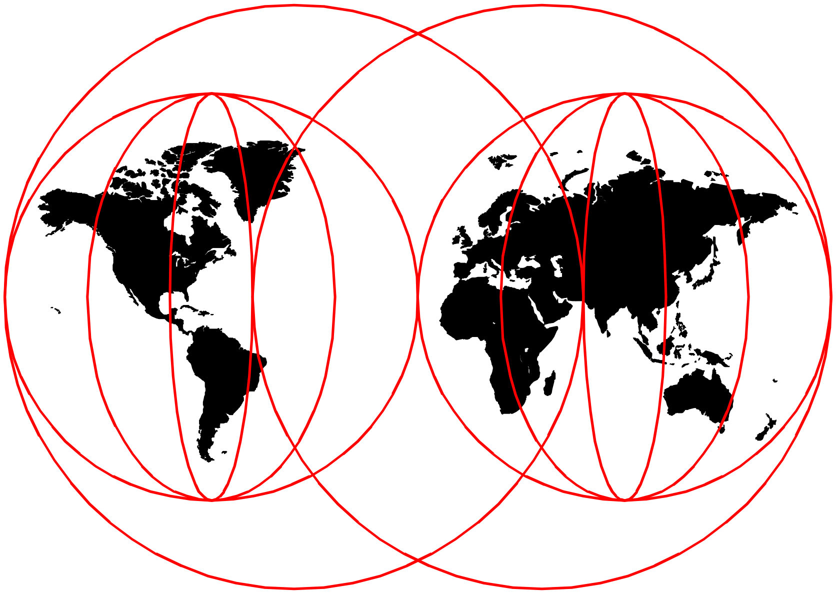
AIX Logical Volume Manager, from A to Z: Introduction and Concepts

Laurent Vanel, Ronald van der Knaap, Dugald Foreman,
Keigo Matsubara, Antony Steel



International Technical Support Organization

www.redbooks.ibm.com

SG24-5432-00

SG24-5432-00

International Technical Support Organization

AIX Logical Volume Manager, from A to Z: Introduction and Concepts

January 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix F, “Special notices” on page 391.

First Edition (January 2000)

This edition applies to AIX Version 4.3, Program Number 5765-C34.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

Contents

Preface

LVM: Logical Volume Manager. What is its role in the AIX operating System? How does it perform this function? Its role is to controls disk resources by mapping data between a more simple and flexible logical view of storage space and the actual physical disks

How it performs this function is in a topic vast enough to fill two books. This first volume, *AIX Logical Volume Manager, from A to Z: Introduction and Concepts* describes the basic components and defines physical volumes, including volume groups and logical volumes. This book also describes the advanced functions of the Logical Volume Manager, such as the mirroring to increase the availability of the data, the stripping to spread data across several disks and increase the performance in accessing large amounts of sequential data, or the online backup function included in AIX Version 4.3.3.

This book is aimed at every IT specialist who wants to know more about the core element of AIX, which is the Logical Volume Manager.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Laurent Vanel is an AIX and RS/6000 specialist at the International Technical Support Organization, Austin Center. Before joining the ITSO three years ago, Laurent Vanel was working in the French RISC System/6000 Technical Center in Paris where he was conducting benchmarks and presentations for the AIX and RS/6000 solutions.

Ronald van der Knaap is a Senior I/T Specialist in the Netherlands. He has 11 years of experience in the Unix/AIX field. His areas of expertise include a wide range of System and Network Management products, AIX related products, HACMP, Performance & Tuning, and RS/6000 SP systems. He has written extensively on LVM Performance and journaled file systems.

Dugald Foreman is an AIX support specialist in England. He has two years of experience in AIX, both spent working for IBM. His areas of expertise include problem determination in software development and the AIX base operating system. He has written extensively on LVM recovery procedures.

Keigo Matsubara is an Advisory I/T Specialist in Japan. He has seven years of experience in the AIX filed. His areas of expertise include a wide range of

AIX related products, particularly RS/6000 SP and high-end storage systems. He has written extensively on Mirroring, Striping, and Concurrent access volume groups. This is his second redbook.

Antony Steel is an Advisory IT Specialist in Australia. He has eight years of experience in the field of Unix. He holds a degree in Theoretical Chemistry from the University of Sydney. His areas of expertise include system performance and customisation, scripting, and high availability.

Thanks to the following people for their invaluable contributions to this project:

Gerald McBrearty
LVM developer

Ram Pandiri
LVM developer

Johnny Shieh
LVM developer

Mathew Accapadi
AIX performance engineer

Mike Wortman
AIX file system developer

Carol Sherman
Graphic designer

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks evaluation” on page 407 to the fax number shown on the form.
- Use the online evaluation form found at `http://www.redbooks.ibm.com/`
- Send your comments in an Internet note to `redbook@us.ibm.com`

Chapter 1. Components of the logical volume manager

This chapter 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. 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 1 illustrates the layout of those components.

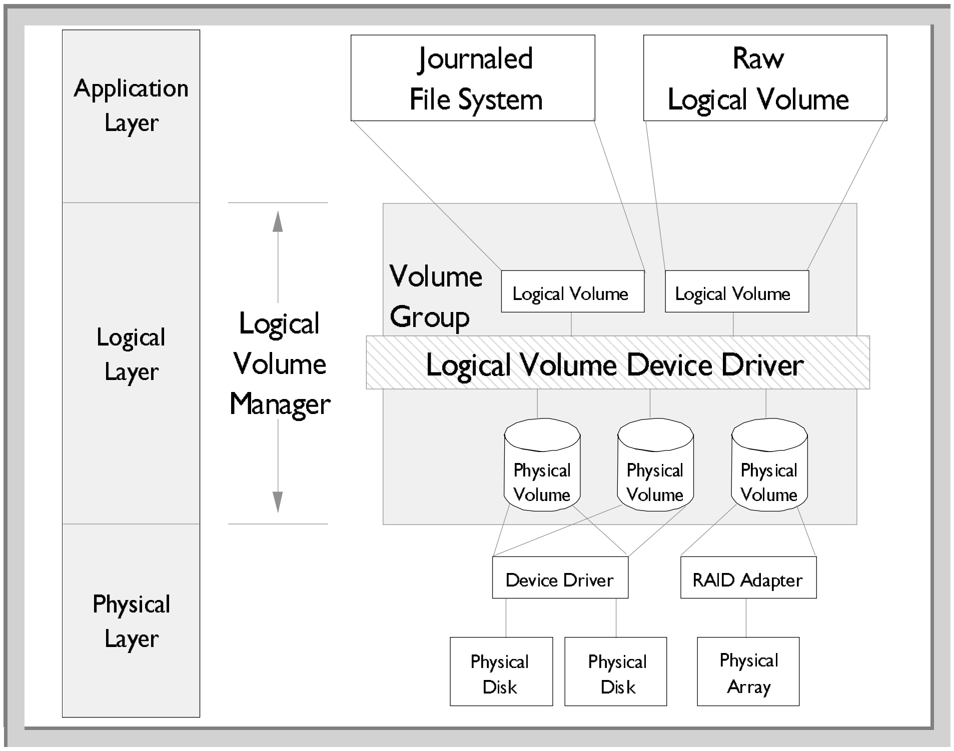


Figure 1. How the LVM fits between the Application Layer and the Physical Layer

1.1 Overview

A hierarchy of structures is used to manage fixed disk storage, and there is a clearly defined relationship (mapping) between them (See Figure 2, which shows the relationship between Volume Groups (datavg), Logical Volumes (lv04 and mirrlv), Logical Partitions (LP1,...), Physical Volumes (hdisk9), and Physical Partitions (PP8,...)).

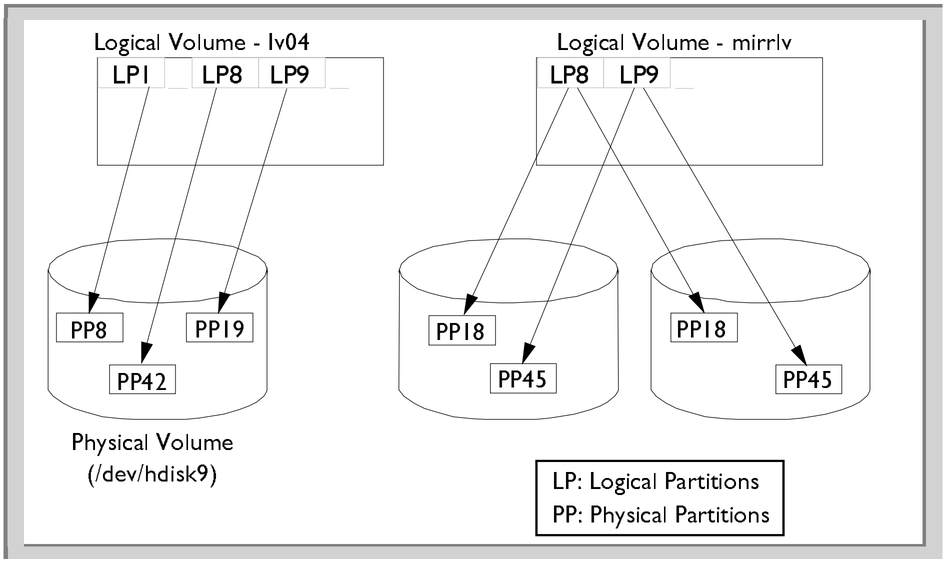


Figure 2. The relationship between the various components of the LVM

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 (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 3). 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.

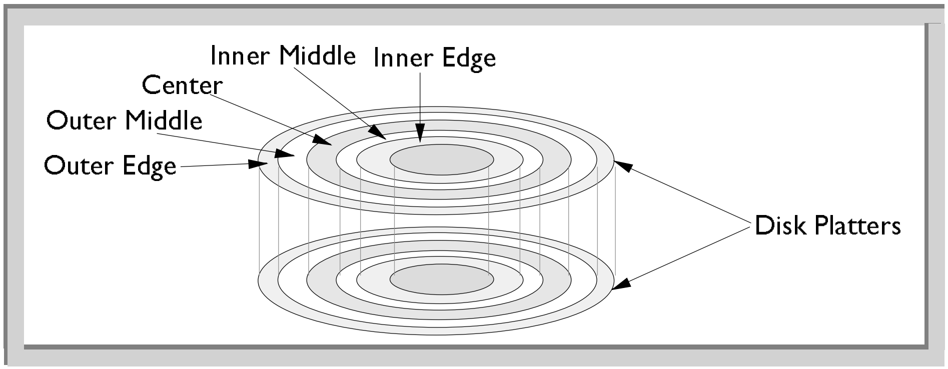


Figure 3. Physical volume regions

The logical volume manager handles a connected Redundant Array of Independent Disk (RAID) disk array in the same manner. The RAID arrary is treated as one disk even though, in most cases, it would be a very large one (see 1.2.1.2, “Definition of RAID” on page 9).

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.

Attention

Mirroring logical volumes on the same physical volumes does not improve availability of data and has only limited application by slightly improving I/O performance. With multiple copies of the same information on the disk, you increase the chance for the disk head to be close to the correct sector.

Although logical partitions are numbered consecutively, the underlying physical partitions are not necessarily either consecutive nor contiguous (see Figure 2).

Logical volumes can be used in a number of ways:

Unstructured — These are called raw logical volumes and are used by the system for a number of purposes, for example paging, and dump spaces. Applications, particularly databases, can also use raw logical volumes, and, in this case, the application manages how the data is stored.

Structured — If the Logical Volume is to be used to hold ordinary files (data or programmes), AIX supplies the Journaled File System (JFS) a hierarchical structure of files and directories to provide the structure.

Logical volumes can be created/modified using commands or the menu driven System Management Interface Tool (SMIT).

Note

While the Web System Management tool will also allow much of the configuration discussed in this book, we will limit this discussion to the use of the actual commands ("command line") and SMIT.

Logical volumes reside only within a volume group. A logical volume can:

- Reside on one physical volume
- Span multiple physical volumes within the volume group
- Have mirrored copies on different physical volumes within the volume group

So far, we have seen the following components:

Physical volumes A storage device that are divided into physical partitions.

Volume groups A collection of one or more physical volumes, independent of type.

Logical volumes A collection of logical partitions, each of which can be mapped to any physical partition in the volume group. If mirroring is used, the logical partition is mapped to two or three physical partitions.

Logical volume manager Controls all this through the logical volume device driver. It takes a complex structure of physical partitions, including mirroring, and presents a simple logical partitions structure to the user/application.

To gain a better understanding of the LVM, we will look at its components in greater detail.

1.2 The logical volume storage concepts

In Figure 2, we saw the relationship between the basic logical storage concepts. We will now look at the following in more detail:

- Physical volumes
- Volume groups
- Logical volumes
- Mirroring

- Striping
- The Object Database Manager (ODM)

1.2.1 Physical volumes

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.

The following are terms used when discussing DASD volumes:

Block A contiguous, 512-byte region of a physical volume that corresponds in size to a DASD sector. This is often used interchangeably with sector.

Partition A set of blocks (with sequential cylinder, head, and sector numbers) contained within a single physical volume.

A physical volume is a DASD structured for requests at the physical level, that is, the level at which a processing unit can request device-independent operations on a physical sector address basis. A physical volume is composed of the following:

- A device-dependent reserved area.
- A variable number of physical blocks that serve as DASD descriptors.
- A number of partitions, each containing a fixed number of physical blocks.

Typical operations at the physical level are read-physical-block and write-physical-block.

The number of blocks in a partition, as well as the number of partitions in a given physical volume, are fixed when the physical volume is installed in a volume group. Every physical volume in a volume group has exactly the same partition size.

A disk must be designated as a physical volume and put in an available state before it can be assigned to a volume group. A physical volume has certain configuration and identification data written on it. This is usually done before the disk is shipped or can be done by running the low level diagnostic format command. This information includes a physical volume identifier (PVID) which is unique to the system. When a disk become a physical volume, it is divided into 512 byte physical sectors. You designate a disk as a physical volume with the `mkdev` or `chdev` commands or by using the System Management Interface Tool (SMIT).

1.2.1.1 The structure of a physical volume

Figure 4 shows the structure of a physical volume. The technology used in the construction of the read/write heads, as well as the composition of the magnetic material used on the platter surface, defines how much information can be stored on the disk device and how long it will take to store/retrieve.

The following terms are used when discussing disk operations:

Sector An addressable section of a track used to record one block of contiguous data

Track A circular path on the both surfaces of a disk platter on which data is written and read. It is a contiguous set of sections that can be swept by the read/write head without the head being moved.

Head This is the single unit that is used to read and write the data on the surface of the platter. There are a number of heads (one for each platter) that are moved as a single unit. This unit is called the actuator.

Cylinder A vertically aligned collection of tracks that can be read without moving the head unit. If the disk consists of n vertically aligned heads, a cylinder consists of n vertically aligned tracks.

Platter A single disk that is coated on one or both sides with material that can be magnetized/de-magnetized by the read/write head.

Region As described above, the physical volume is divided into five regions for allocation purposes.

The earliest read/write heads used small coils that generated a magnetic field when a current was passed through them, thereby, changing the magnetic properties of a section on the surface of the disk. This magnetic polarity would define the binary value stored. By passing a coil over the surface of the disk and causing it to intersect with the magnetic field of each area, a small current would be generated in the coil; its direction dependent on the polarity of that section of disk. The newer technologies in read/write heads and greater densities in the packing of information on the disk surface have lead to higher overall capacity disk drives.

There have also been changes in technology. Some systems now use a different technology for the read heads called magneto resistive (MR). These read heads sense the variations in the electrical resistance of the platter surface rather than relying on the electrical induction.

The greatest amount of time in the whole process of reading/writing data on a physical volume is spent in locating the information and returning it to the requesting task. In the simplest case, a request arrives for data, the head is moved to the correct position, the required number of blocks are read, and the data returned. The next request arrives, the head is repositioned and so on. This process involves a significant delay. In order to minimize this, some changes have been introduced:

Elevator sorting — Incoming requests are sorted by location so that the actuator can fulfill more requests in one pass, thus, minimizing seek time

Read ahead — Data is read in after the initial request and cached, thus, making the assumption that the next read request from the given process will probably want the next portion of data on the disk. Thus, the next request for data can be quickly fulfilled, as it is coming from the cache and not relying on the disk to retrieve it.

Banding — As data is written at a fixed rate, there will be larger gaps between the data the further out from the center of the disk one is. Banding partitions the disk into sections of increased data density as the head moves out from the center. This ensures even data density and an increase in the overall capacity of the disk. It also results in higher performance read and writes on the outer partitions of the disk due to the higher data density.

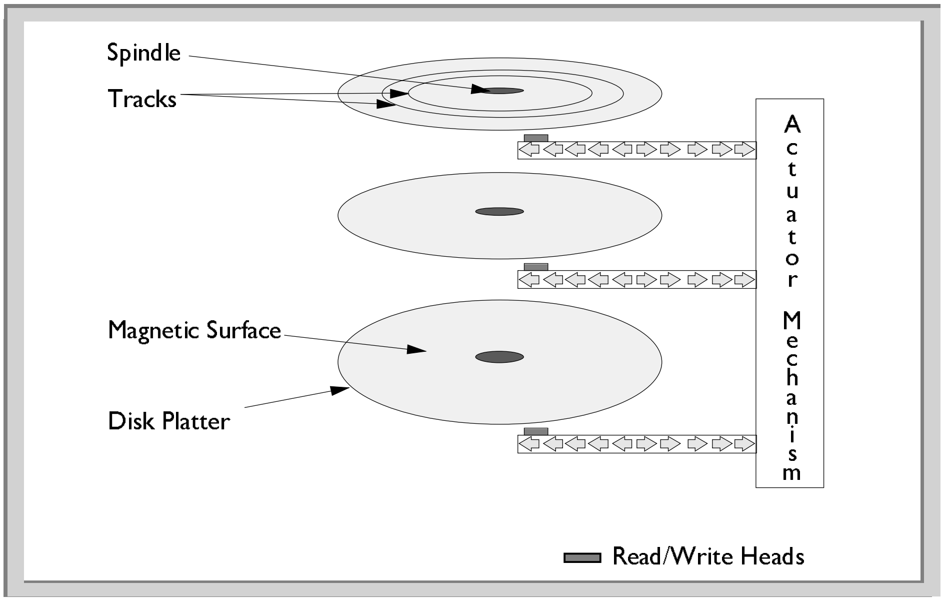


Figure 4. The structure of a disk drive

1.2.1.2 Definition of RAID

Another method of improving access to data was by introducing the ability to spread I/O simultaneously across multiple disks. This functionality is provided through Redundant Array of Independent Disks (RAID) support. RAID is defined with a number of levels:

RAID 0 This is also known as data striping. This is when the data is split into smaller chunks (strips) and spread over a number of physical volumes, thus, each read is conducted from a number of physical volumes in parallel. Data transfer rates are higher for sequential operations, as I/O operations can be overlapped, and higher for random access, as the data is better scattered across multiple physical volumes. RAID 0 is designed for performance not availability.

RAID 1 This is also known as disk mirroring. In this case, duplicate copies of data are kept on separate physical volumes. In most instances, each physical volume will have at least one other physical volume with an exact copy of its data. As discussed in Chapter 7, “Performance” on page 303, this is primarily for availability but, in some cases, may improve performance. The drawback of RAID 1 is that the number of physical volumes must be doubled.

RAID 2/3[1] RAID 2 and 3 describe disk arrays that are 'parallel process' arrays, where all the drives in the array operate in unison. This is similar to striping, but data is split into chunks, and each chunk is written to the same physical location on different physical volumes. Instead of mirroring the data, parity information is calculated for each chunk of data and written separately. For RAID 3, the parity data can only be stored on one physical volume, while with RAID 2, this can be spread over a number of physical volumes. Performance of RAID 2/3 is good for large amounts of data but poor for small requests, as their is no overlapping or independent operation.

RAID 4[1] This configuration overcomes some of the disadvantages of RAID 3 by striping the data over all physical volumes but one. This last physical volume is used for the parity data. This configuration is good for large reads, but the parity disk becomes a bottleneck when writing.

RAID 5 This configuration overcomes the bottleneck in RAID 4 of having only one physical volume for the parity data by spreading all the parity data across all disks. In a disk array, each chunk of data strip is on n-1 disks, with the parity information on the nth disk. The parity data is never stored on the same disk of the data it protects.

[1] These forms of RAID are not used in AIX

AIX Version 4.3.3 supports

- RAID 0
- RAID 1
- RAID 0 and RAID 1 - not defined together as a standard

Disk subsystems that support RAID at the time of publication include:

- IBM 7135 RAIDiant array
- IBM 3514 High Availability External Disk Array
- IBM 9570 Disk Array Subsystem
- SSA Adapters (6217(4-I), 6218(4-J), 6219(4-M), 6215(4-N), 6225(4-P)) support RAID

But it is important to check with IBM locally to get information on the latest hardware and microcode levels required for RAID.

1.2.1.3 Factors when considering physical storage options

In this section, we will look at the capabilities and functions of the different storage options to provide an understanding of the rationale behind some of the storage management policies.

As usual, it is a balancing act between requirements (determined by application, users, and physical environment) and cost.

Adapters

There are some important things to consider when looking at storage technology, starting with the adapter:

- Cabling requirements (lengths, adapter, disk, adapter to adapter).
- Performance / Reliability (maximum sustained and burst data transfer rates and reliability will affect performance).
- Addressability - How many devices can be attached per adapter and if multiple systems can be used.
- Device support - Use the type of devices you need.
- Cost - Both adapter and devices.

There are four main types of adapters:

Small Computer Systems Interface (SCSI)

This is the most common type of adapter. The original standard (now known as SCSI-1) allows up to seven devices on a 1 byte parallel arbitrated bus. Since then, there have been SCSI-2, Fast SCSI, Wide SCSI, Fast/Wide, and Differential SCSI, all of which have offered improvements in transfer rates, cable lengths and availability. Backward capability has been maintained. SCSI supports multiple adapters on the bus.

High Performance Parallel Interface Adapter (HiPPI)

This allows point-to-point communications up to 800 Mb/sec to a maximum distance of 25 m.

ESCON Channel Adapter

Allows 17 MB/s over optical fibre links up to 20 km.

Serial Storage Architecture

Serial Storage Architecture, or SSA, is an emerging standard defining a new connection mechanism for peripheral devices. The architecture specifies a serial interface that has the benefits of more compact cables and connectors, higher performance and reliability, and, ultimately, a lower subsystem cost. A general purpose transport layer provides for 20 MB per second full duplex communications over 10 meter copper cables (extended with use of optical fibre). Devices are connected together in loops with up to 128 nodes (devices) allowed per loop. Information is transmitted in 128 byte frames that are multiplexed to allow concurrent operations. In addition, the full duplex communications allows simultaneous reading and writing of data.

Fibre Channel arbitrated loop (FC-AL)
This a new set of interfaces and protocols that intends to support linking storage devices using high-speed network communications hardware. FC-AL has standards supported by a trade association in which IBM, Hewlett-Packard, Sun, and others are members. FC-AL allows up to 100 MB/s around a loop of up to 126 devices.

Disk Storage

All disks have a fairly similar design (see Figure 3 on page 3). They have a number of platters fixed on a central hub, which is rotated at high speed. Both surfaces of each platter are coated with a thin film of magnetic material on which the data is stored. The read/write heads are fixed on the end of arms (actuators). To move to different tracks, the heads are moved back and forth by the actuators as one unit.

The following terms are used in discussing disk performances:

Seek time The time it takes the actuator to move head to the correct track

Rotational latency The time it takes for the platter to rotate to bring the correct sector under the head.

As discussed previously, the density of the data on the disk is determined by the technology of the head and the film on the platter surface.

The following factors are key in determining the performance of the storage system. More details will be discussed in 7.2, “The physical layer” on page 305.

- Input/output ratio (the quantity of code that is executed per sector of data)
- The number of disks that contain data
- Disk drive performance
- Disk drive data path performance
- Size of blocks accessed (does the application use large or small blocks of data?)

- Pattern of data (random, sequential)
- Ability to cache/ read ahead
- Response time required.

Availability and fault tolerance
The need for availability of the system will also influence the decision on which storage technology to use and will also influence the performance of the I/O subsystem.

1.2.2 Disk Independence

One great feature of the LVM is its ability to support a mixture of disks in a volume group, regardless of their type or location. Thus SSA, Serial, SCSI, and RAID drives can make up one volume group and may reside across a number of adapters.

The basic design of the interaction between the LVM and the disk device driver always ensures that the LVM's use of the physical volume will have the same behavior, regardless of the type of physical volume being used in the LVM. Thus, a physical volume, such as a SSA (Serial Storage Architecture) disk, behaves the same in the LVM as a standard SCSI drive, although they use different adapter and device drivers.

The LVM concentrates on the special file that represents the physical volume. Although it is not required by the LVM, almost all physical volumes are represented as `hdiskx`. The only exceptions to the transparent use of disks by the LVM are the read/write optical disk drives. There is special code in the high level LVM commands that checks for the existence of optical devices. Because of the removability of optical media, there is special handling to ensure that the LVM knows the true identity of the optical disk drive.

This not only presents a simple view of data storage to the application, but allows the system administrator to better match data needs to the storage technology without adding new levels of confusion.

1.2.3 The physical volume identifier (PVID)

Sometime, it can get confusing as to the true name of a physical volume and how it is recognized by the system. This usually occurs during service calls or after a system has crashed. The ODM is used to attach an `hdiskx` name to a physical volume ID (PVID) of a disk. The PVID is the soft "serial number" of a disk that is created, usually just once during the lifetime of a physical volume's usage on AIX. It may need to be recreated using the `dd` command if someone accidently overwrites it, or uses the low level diagnostics to reformat the disk.

To make a disk into a physical volume, the PVID is placed onto the disk. The PVID is an combination of the machine’s serial number (from the systems EPROMs) and the date the PVID was generated. This combination ensures the extremely low chance of PVIDs being duplicated. When the system is booted, the disk configurator looks at the PVID residing on the disk and compares it with an entry in the ODM. If an entry is found, then the disk is given the `hdiskx` number in the ODM that is associated with the PVID. If there is no matching entry, then the next name in the pool of ’free’ `hdisk` names is allocated to the physical volume.

If an `hdisk` is removed with the `rmdev -l hdiskx -d` command, then the `hdisk` name is available for use by another disk.

The PVID can be obtained by issuing:

`/usr/sbin/lpquerypv  -H /dev/hdiskx` (AIX Version 4 and above)

`/usr/bin/hdf` (AIX Version 3.2.x)

Since the PVID is contained in the ODM, there can be times when the ODM no longer matches the ’true’ PVID on the disk.

There are instances, such as multi-host environments, where you may wish to change this name so that disk numbers are consistent over multiple hosts. This involves a simple procedure of creating ’dummy’ disks (using `mkdev`) so that when the real disks are assigned physical volume numbers by the configuration manager, the `hdisk` numbers will match over the different systems. More details can be found in the *HACMP Cookbook*, SG24-4553.

1.2.4 Physical Volume Layout

A physical volume consists of a logically contiguous string of physical sectors. Sectors are numbered 0 through the last physical sector number (LPSN) on the physical volume. The total number of physical sectors on a physical volume is LPSN + 1. The actual physical location and physical order of the sectors are transparent to the sector numbering scheme.

Note

Sector numbering applies to user-accessible data sectors only. Spare sectors and Customer-Engineer (CE) sectors are not included. CE sectors are reserved for use by diagnostic test routines or microcode.

The first 128 sectors of a physical volume are reserved to store various types of information for its configuration and operation. Table 1 describes what is stored in these sectors. Further detail is contained in the header file: `/usr/include/sys/hd_psn.h`

Table 1. The structure of the first 128 sectors of a physical volume

Starting Sector	**Record**	**Description**
0	bootrecord	This is one sector that contains information that allows the Read Only Storage (ROS) to boot the system. The physical volume identifier is also contained in the boot record. A description of what is contained in the boot record can be found in `/usr/include/sys/bootrecord.h`
1	Configuration	
2	MWC cache and alternate cache	The mirror write consistency (MWC) records must stay contiguous. They identify which logical partitions may be inconsistent if the system is not shutdown properly. When the volume group is varied back online for use, this information is then used to make logical partitions consistent again. The MWC Caches are allocated two sectors, but the alternate cache doesn't start at the beginning of the second sector.
7	LVM	This sector is used by the LVM when a physical volume is part of a volume group. It contains LVM ID field, size of the VGDA, and the location (at the end of the physical volume) that is reserved for the relocation of bad blocks. More information can be found in: `/usr/include/lrmrec.h`
8	bad block	This record points to the bad block directory, which is used to record the blocks of the physical volume that have been diagnosed as unusable. Each record consists of a reason why the block has been marked bad, the logical sector number of the block, and the logical sector number of the relocated block. Details can be found in: `/usr/include/sys/bbdir.h`.
22	bad block length	The size of the bad block directory.

Starting Sector	Record	Description
64	config backup	A backup copy of the configuration record.
70	LVM backup	A backup of the LVM record.
71	bad block backup	Backup of the bad block directory.
120	concurrent config work directory	The concurrent configuration scratch sectors are used by each physical volume when a configuration operation is sent across.
128	first non reserved	The volume group descriptor area and status area (VGDA / VGSA) record starts here. There is further discussion of this area under the section on volume groups.

1.2.4.1 Physical Partitions

When you add a physical volume to a volume group, the physical volume is partitioned into contiguous, equal sized units of space called physical partitions. A physical partition is the smallest unit of storage space allocated. Physical volumes inherit the volume group's physical partition size, which can only be set when the volume group is created (for example, using `mkvg -s`).

In pre-AIX 4.3.1 versions, if you add a new volume group, the physical partition sizes in megabytes you are allowed to chose are: 1, 2, 4, 8, 16, 32, 64, 128, and 256.

The support for physical partition sizes of 512 MB and 1024 MB have been added to AIX 4.3.1. Hence, if you add a new volume group, the physical partition sizes in megabytes you are allowed to chose are: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, and 1024 MBs.

Note

If you select a physical partition size of 512 or 1024 MBs on a AIX 4.3.1 or later system, the volume group created cannot be imported on older versions of AIX.

1.2.5 Maximum number of physical partitions per physical volume

In the design of the LVM, each logical partition maps to one or more physical partitions, up to a maximum of three. Each physical partition maps to a number of disk sectors. By default, the design of the LVM limits the number of Physical Partitions that the LVM can track to 1016. Any physical volume created with a PP size that resulted in more than 1016 physical partitions would have the remaining PPs not tracked by the VGSA.

For RAID systems, the LVM is unaware of the size of the individual disks that make up /dev/hdiskx. The LVM bases the 1016 limitation on the AIX recognized size of /dev/hdiskx and not the real physical disks that make up /dev/hdiskx.

In AIX Versions 4.1.2 and below, users were allowed to create physical volumes that violated this limit. However, only the first 1016 physical partitions would be monitored if they were stale or not.

In AIX Version 4.3.2 and 4.3.3, new improvements are offered that allow the user to create volume groups with up to 128 disks and have a maximum of 1024*1016 partitions in a volume group. This enhancement called big VGDA, is discussed in 1.2.7.5, “Big VGDA or VGDA expansion” on page 30.

Table 2 shows the minimum partition size for each physical volume size based on the 1016 limitation. These are also the default partition size when a logical volume is created.

Table 2. Default PP sizes with 1016 physical partitions per disk

Default physical partition size (MB)	Physical volume size
2	< 300 MB
4	< 4 GB
8	< 8 GB
16	< 16 GB
32	< 32 GB

In AIX Version 4.3.1, the option was added to allow users to create physical volumes with more than 1016 physical partitions.

1.2.6 Increasing number of physical partitions per physical volume

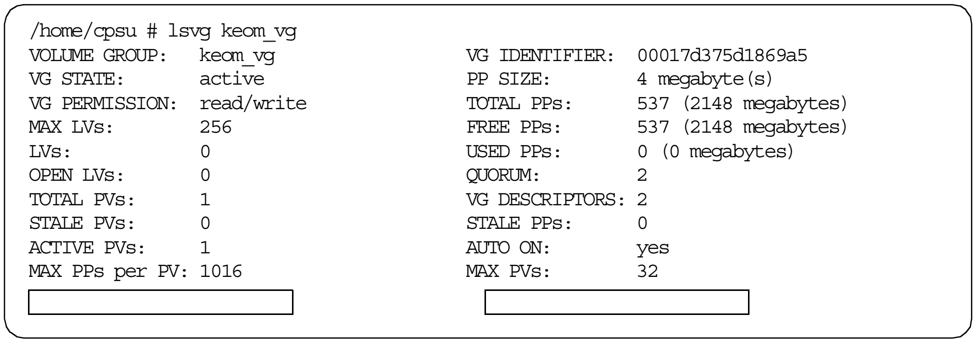
In version 4.3.1, an option was added to allow the user to create physical volumes whose physical partitions were limited by a multiple of 1016 using `chvg -t n`, where n is the multiple of 1016. This will, however, reduce the number of physical volumes that are allowed in the volume group. This is because each volume group is limited to 32 512 physical partitions, which is 32 physical volumes, each with 1016 partitions. Thus, if the number of physical partitions factor is set to 4 (4064 partitions allowed per physical volume), then only eight physical volumes will be allowed in the volume group.

Note

After the `chvg -t` command has been run, and if you need to add more disks to the volume group, you can change back to 1016 partitions a disk if you only have disks that need 1016 or less partitions in the volume group. This volume group cannot be imported into systems running AIX 4.3.0 or below.

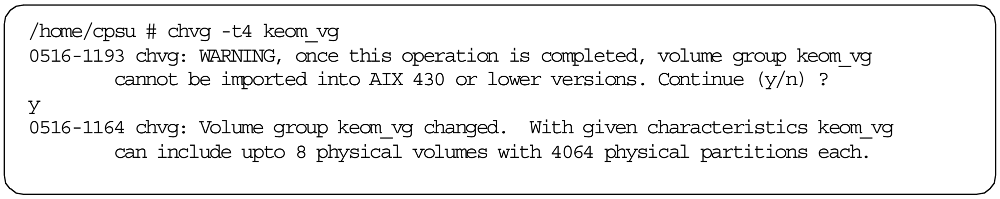
The screens below show the result of using a factor of 4.

A test volume group.



```
/home/cpsu # lsvg keom_vg
VOLUME GROUP:   keom_vg                  VG IDENTIFIER:  00017d375d1869a5
VG STATE:       active                   PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      537 (2148 megabytes)
MAX LVs:        256                      FREE PPs:       537 (2148 megabytes)
LVs:            0                        USED PPs:       0 (0 megabytes)
OPEN LVs:       0                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     1                        AUTO ON:        yes
MAX PPs per PV: 1016                     MAX PVs:        32
```

The factor set to 4.



```
/home/cpsu # chvg -t4 keom_vg
0516-1193 chvg: WARNING, once this operation is completed, volume group keom_vg
        cannot be imported into AIX 430 or lower versions. Continue (y/n) ?
y
0516-1164 chvg: Volume group keom_vg changed.  With given characteristics keom_vg
        can include upto 8 physical volumes with 4064 physical partitions each.
```

```
The volume group changed.
```

```
/home/cpsu # lsvg keom_vg
VOLUME GROUP:   keom_vg              VG IDENTIFIER:  00017d375d1869a5
VG STATE:       active               PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write           TOTAL PPs:      537 (2148 megabytes)
MAX LVs:        256                  FREE PPs:       537 (2148 megabytes)
LVs:            0                    USED PPs:       0 (0 megabytes)
OPEN LVs:       0                    QUORUM:         2
TOTAL PVs:      1                    VG DESCRIPTORS: 2
STALE PVs:      0                    STALE PPs:      0
ACTIVE PVs:     1                    AUTO ON:        yes
MAX PPs per PV: 4064                 MAX PVs:        8
```

Now, using this factor, those physical volumes created in earlier versions of AIX that violated the 1016 limit can be brought back into a supported state.

This command cannot be run if there are any stale physical partitions in the volume group. The `-t` flag may also not be used if the volume group is varied on in concurrent mode.

Note

The `-t` flag may require a modification of the LVM meta data. In this situation it is advisable to vary off the volume group in management mode to ensure the integrity of the volume group.

Table 3 gives details of the effect of changing the `-t` factor on the maximum partitions per physical volume and the maximum number of physical volumes for the volume group.

Table 3. Effect of -t factor in chvg

chvg -t factor	Maximum partitions / PV	Maximum disks / VG
1 (default size)	1016	32
2	2032	16
3	3048	10
4	4064	8
5	5080	6
6	6096	5
7	7112	4
8	8128	4

chvg -t factor	Maximum partitions / PV	Maximum disks / VG
9	9144	3
10	10160	2
16	16256	2

1.2.6.1 Querying a physical volume

The `lspv` command is useful in querying the usage of physical partitions on a physical volume.

The `lspv hdiskx` command will return high level information about the physical volume, the PVID, VGID, volume group name, number of VGDAs, number of logical volumes, and the physical partition usage by region.

```
/home/red # lspv hdisk11
PHYSICAL VOLUME:    hdisk11                  VOLUME GROUP:     keovg
PV IDENTIFIER:      00017d37e671fe4b         VG IDENTIFIER     00017d377005016c
PV STATE:           active
STALE PARTITIONS:   0                        ALLOCATABLE:      yes
PP SIZE:            4 megabyte(s)            LOGICAL VOLUMES:  3
TOTAL PPs:          535 (2140 megabytes)     VG DESCRIPTORS:   2
FREE PPs:           498 (1992 megabytes)
USED PPs:           37 (148 megabytes)
FREE DISTRIBUTION:  107..70..107..107..107
USED DISTRIBUTION:  00..37..00..00..00
```

The `lspv -l hdiskx` command will show the allocation of physical partitions for each logical volume on the physical volume by region.

```
/home/red # lspv -l hdisk11
hdisk11:
LV NAME               LPs   PPs   DISTRIBUTION          MOUNT POINT
ulv01                 12    12    00..12..00..00..00    N/A
ulv02                 24    24    00..24..00..00..00    /home/kim
loglv00               1     1     00..01..00..00..00    N/A
```

The `lspv -p hdiskx` command will show the allocation of physical partitions by region.

```
/home/red # lspv -p hdisk11
hdisk11:
PP RANGE  STATE   REGION        LV ID               TYPE       MOUNT POINT
  1-107   free    outer edge
108-119   used    outer middle  ulv01               jfs        N/A
120-143   used    outer middle  ulv02               jfs        /home/kim
144-144   used    outer middle  loglv00             jfslog     N/A
145-214   free    outer middle
215-321   free    center
322-428   free    inner middle
429-535   free    inner edge
```

The `lspv -M hdiskx` command will show the logical partition to physical partition map for each logical volume.

```
/home/red # lspv -M hdisk11
hdisk11:1-107
hdisk11:108     ulv01:1
hdisk11:109     ulv01:2
hdisk11:110     ulv01:3
hdisk11:111     ulv01:4
hdisk11:112     ulv01:5
hdisk11:113     ulv01:6
hdisk11:114     ulv01:7
hdisk11:115     ulv01:8
hdisk11:116     ulv01:9
hdisk11:117     ulv01:10
hdisk11:118     ulv01:11
hdisk11:119     ulv01:12
hdisk11:120     ulv02:1
hdisk11:121     ulv02:2
    .................
hdisk11:142     ulv02:23
hdisk11:143     ulv02:24
hdisk11:144     loglv00:1
hdisk11:145-535
```

The `lslv -p hdiskx` command will show the status of each physical partition for the physical volume. This is shown in the Section 1.2.8 on logical volumes.

1.2.7 Volume groups

As discussed previously, a volume group is a collection of physical volumes. When you install the system, one volume group (the root volume group, called `rootvg`) is created. New volume groups can be created with the `mkvg` command. Physical volumes are added to a volume group using the `extendvg` command and can be removed using the `reducevg` command. It is important to remember that the `rootvg` has attributes that differs from the user defined volume groups, particularly, it cannot be imported or exported.

1.2.7.1 The Volume Group Identifier

Just as the PVID is a soft serial number for the disk, the Volume Group also has an identifier called the Volume Group Identifier (VGID). It is a soft serial number for the volume group and is the number that is referenced by the low-level LVM commands not the volume group name. It also forms the base for all logical volume IDs created for the logical volumes in the volume group.

1.2.7.2 Volume groups and major numbers

When a volume group is created, a major number is assigned. This can be specified (`mkvg -V major_number`) or the next lowest available number will be assigned. The `lvlstmajor` command will show which major numbers are available.

It is recommended in multi-host environments that the major numbers are consistent across the cluster, particularly, if file systems are being cross mounted. This is because the NFS uses the major number as part of the file handle, which is used when remote systems refer to the file.

When a volume group is created, two files are created in /dev with the selected major number:

- `__vgnn` where `nn` is the major number
- `vg_name`

As logical volumes are created, they will have the same major number and consecutive minor numbers starting from 1 (for both the character and block devices). The screen below shows the major and minor numbers for the volume group (`agvg`) and the logical volume (`jonlv`).

```
-> Show what major numbers are free
/dev # lvlstmajor
42,44...

-> Now create a volume group (agvg) with one logical volume (jonlv)

/dev # ls -l | grep 42,

crw-------   1 root     system    42,  0 Sep 20 17:32 __vg42
crw-rw----   1 root     system    42,  0 Sep 20 17:31 agvg
brw-rw----   1 root     system    42,  1 Sep 20 17:32 jonlv
crw-rw----   1 root     system    42,  1 Sep 20 17:32 rjonlv
```

1.2.7.3 Volume group status area (VGSA)

If a physical volume is part of a volume group, it contains two additional reserved areas. One area contains both the volume group status area and the volume group descriptor area and follows the first 128 reserved sectors (see Table 1 on page 15). The other area is at the end of the physical volume and is reserved as a relocation pool for bad blocks that must be software-relocated. Both of these areas are described by the LVM record. The space between these two reserved areas is divided into equal-sized partitions, the physical partitions assigned by the LVM for the user.

The volume group status area contains the state of all allocated physical partitions on all of the physical volumes in the volume group.

The volume group status area (VGSA) is also used when logical volumes are mirrored, and it records information on stale partitions. It is also used by quorum (see below) if the VGSAs differ between physical volumes.

The important information is contained in 127 byte records for each physical volume in the volume group. This record represents the 1016 physical partitions, that is, each byte contains the status of eight partitions. However, if the `chvg/mkvg -t` factor is used to increase the number of physical partitions on the disk, then this area will be larger. This is why these volume groups are not supported by versions of AIX Version 4.3.1 and earlier.

If the mirror copy of a particular physical partition cannot be written, then the appropriate bit in the VGSA is changed. A quick bit mask is then used to determine which physical partitions, if any, have become stale and must undergo a data re-sync when the problem is resolved.

While a physical partition is marked stale, it is not used for a read request or updated with a write request.

A physical partition changes from active to stale if it is no longer consistent with its other copies in the logical partition. This can be caused by a write error or by the physical volume not being available for a write. The physical partition state will change from stale to active after the logical partition is successfully re-synchronized. For this operation, the logical volume device driver reads from an active partition in the logical partition and writes that data to any stale partitions. If this has been successful, then the status of the physical partition is changed to active (the bit in the VGSA set to 0). If an error occurs during this re-synchronization, the partition will remain stale.

Normal input/output operations on this logical partition can continue while the re-synchronization is taking place.

Table 4 on page 25 shows the detailed structure of the VGSA.

1.2.7.4 Volume group descriptor area (VGDA)

The VGDA contains information about the volume group, the logical volumes that reside in the volume group, and the physical volumes that make up the volume group. These VGDAs are also used in quorum voting. The status information pertaining to the physical volume status is limited to active or missing.

As the VGDA contains information about each of the disks in the volume group, only one disk need to be specified when the `importvg` command is being used to import a volume group into a system. The importvg command will go to the specified physical volume, read the VGDA, find out what other physical volumes make up the volume group (by their PVID), and import them into the system and the ODM as well.

As the VGDA contains information about logical volumes in the volume group, any changes to a logical volume (creation, extension, or deletion) will mean that the VGDA on all disks is updated.

The volume group descriptor area (VGDA) is divided into the following:

- The volume group header.

  This header contains general information about the volume group and a time stamp used to verify the consistency of the VGDA.

- A list of logical volume entries.

  The logical volume entries describe the states and policies of logical volumes. This list defines the maximum number of logical volumes allowed in the volume group.

- A list of physical volume entries.

  The physical volume entries describe the number of physical partitions, the state of the physical volume, and the physical partition map. The size of the physical volume list is variable because the number of entries in the partition map can vary for each physical volume. For example, a 200 MB physical volume with a partition size of 1 MB has 200 partition map entries.

- A volume group trailer.

  This trailer contains an ending time stamp for the volume group descriptor area.

Table 4 shows the detailed structure of the VGDA.

Table 4. Structure of the VGSA / VGDA

Position	Field	Description
128	VGSA	
		Time stamp at the beginning of the VGSA
		Physical volume missing flag
		Stale map - 127 * vgsa factor 8bit records - 1 record for 8 physical partitions. There is one map for each physical volume in the volume group. One for each physical volume
		vgsa factor - for more than 1016 partitions[1]
		VGSA Version
		Reserved
		Time stamp at the end of the vgsa (consistency check)
136 (384[1])	vgda header	
		First timestamp for consistency checking
		vgid
		Number of logical volumes
		Maximum number of logical volumes - 128 (512[1]) default
		PP size as x, where 2^x is size (range 20-30)
		Number of Physical Volumes

Position	Field	Description
		Number of Volume Group Descriptor Areas
		vgda size - 2098(8242[1])
		bigvgda flag[1]
		Quorum on/off[1]
		autovaryon yes/no[1]
		vgda checksum[1]
		Reserved
For each logical Volume	lv header	
		Logical number (0 ... to number of logical volumes - 1)
		Pointer to name list
		Logical volume maximum size
		Logical volume state (open/closed sync/stale)
		Number of mirror copies
		Mirror write policy
		Number of logical partitions
		read/write flag
		Bad block relocation desired
		Write verify
		MWC flag
		Stripe flag
		Stripe width
		Information on the special files permission[1]

Position	Field	Description
		Logical volume control block[1]
		Reserved
For each PV	physical volume header	Physical volume number
		Physical volume ID
		Number of physical partitions
		Physical volume state (active/missing)
		Number of vgdas on disk
		Map of Logical volume to physical partition
		Reserved
Name list		Mapping of logical volume numbers to name
vgda trailer		Concurrency flag
		2nd timestamp for consistency checking
2242 (8906[1])		2nd VGDA (if appropriate) - same structure as first vgda

[1] For a big VGDA

The variable size of the physical partition map is the reason that some volume groups will not achieve their default maximum number of physical volumes, particularly, those consisting of large physical volumes.

Every physical volume has two VGDAs, which are kept up to date, but they are only used as follows:

- One physical volume - Both copies used
- Two physical volumes - Both VGDAs used on the first physical volume, the first VGDA on the second physical volume. The first physical volume is the first one in the volume group. Either the volume group was initially created

with one physical volume, or the order in which the physical volumes were listed during the creation.

- Three or more physical volumes - The first VGDA from each physical volume is used.

Let us look at what happens in a simple case with two physical volumes. Initially, we have a volume group (datavg) consisting of one physical volume (hdisk4) and one logical volume (ulv00). There will be two copies of the VGDA on hdisk4.

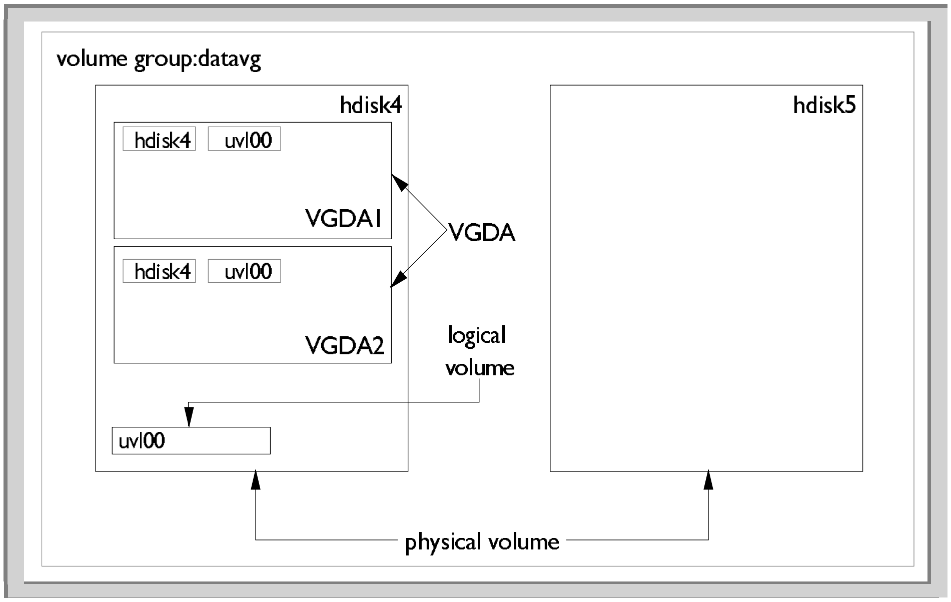


Figure 5. The simple case of one disk in the volume group with one logical volume

We now add another physical volume (hdisk5) to the volume group, and the VGDA is updated, and two VGDAs will be created on the second physical volume. To avoid confusion, we will only look at the first VGDA on the second physical volume, as the second one is not used by the logical volume manager.

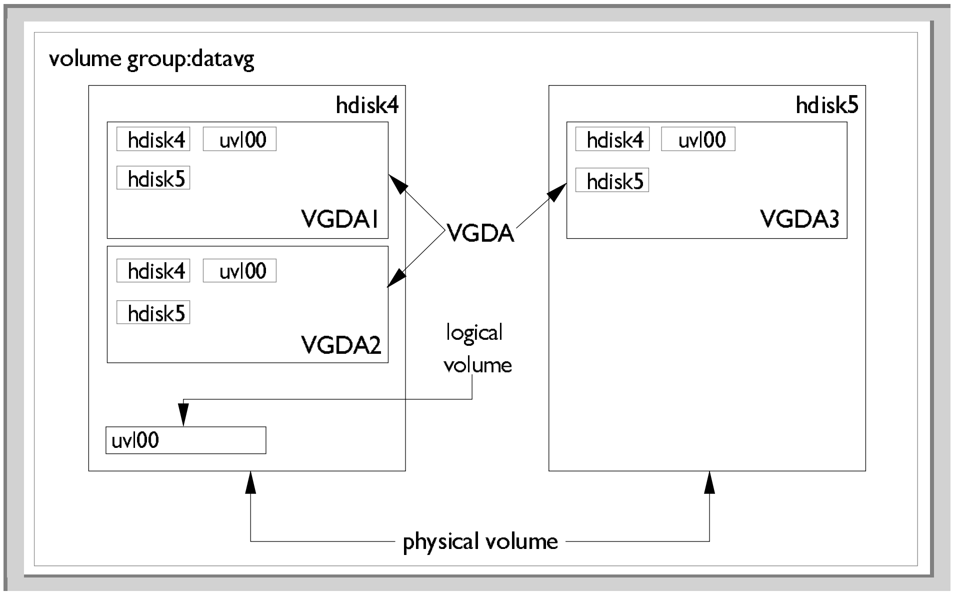


Figure 6. A second disk is added and the VGDA is updated

Two further logical volumes are added to the volume group (ulv01 and ulv02). The VGDAs are all updated.

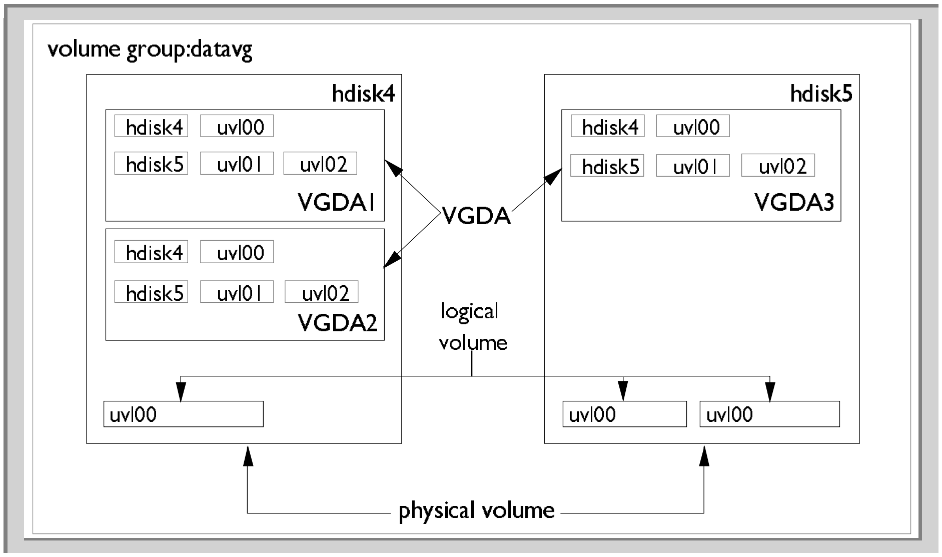


Figure 7. Two further logical volumes added

1.2.7.5 Big VGDA or VGDA expansion

The original limitation to LVM of 32 disks per volume group was becoming a problem with systems with a large number of disks. Especially with the increased use of LVM mirroring, the need for larger volume groups grew.

As the result, AIX Version 4.3.2 introduced the -B option. This changes the maximum number of disks that can be included into a volume group from 32 to 128 disks. And, the number of logical volumes that can exist in that volume group has grown from 256 to 512 logical volumes. AIX Version 4.3.3 introduced the -G option, which allows the VGDA to include up to 1016*1024 partitions.

The obvious penalty is that the bigger VGDA is taking up more space on the physical volume; thus, the usable space on each physical volume in the volume group is reduced.

The opportunity has also been taken with the expansion of the VGDA to include some additional information. In particular, if the permissions of the special logical volume files have been changed, then this information can be stored in the VGDA. Thus, when a big VGDA volume group is imported, the modified permissions are preserved.

An important consideration is the movement of physical partitions to handle the expansion in the VGDA. When the VGDA is expanded, the first few physical partitions (the number of partitions depends on the partition size) must be made available. If they are currently allocated for use, they will be migrated to other free partitions on the same physical volume. The physical partitions are then renumbered to reflect the loss of the first few partitions for the bigger VGDA. Hence, the logical to physical partition mappings are also changed. This is particularly important for those sites that have configured specific logical volumes on specific regions of the physical volume. Similarly, if map files are used or backups made with the map option, then they will have to be updated after the VGDA is changed.

The following should be taken into account before expanding the VGDA:

- The -B and -G flags cannot be used if there are any stale partitions or any open logical volumes in the volume group.
- Big VGDA volume groups cannot be imported into AIX Version 4.3.1 and below.
- There must be enough free partitions available on each physical volume to cater for the expansion of the VGDA.

- The -B and -G flags cannot be used if the volume group is going to be concurrent capable.
- Every VGDA operation (creating, updating logical volumes, adding physical volumes, and so on) will take longer, as the VGDA is now bigger.
- Map files must be updated after the change.
- Some logical volumes may have to be repositioned after the change.
- If backups are made with the map option, then it is recommended that backups be made just before and just after the expansion.
- The big VGDA cannot be converted to a normal VGDA without a backup and total restore.

1.2.7.6 Concurrent volume groups

A concurrent volume group is a volume group that is configured on creation as being concurrent capable (`mkvg -c`). If more than one system can access the physical volumes, then, through a concurrent capable volume group, they can now concurrently access the information stored on them. Initially, this was designed for high-availability systems, with the high-availability software controlling the sharing of the configuration data between the systems, but the application must control the concurrent access to the data.

Further information on concurrent capable volume groups is found in Chapter 4, “Concurrent access volume groups” on page 183.

1.2.7.7 Portability of volume groups

One of the great attributes of the LVM is the ability of the user to take the disk or disks that make up a volume group to another RS/6000 system and access the information created by the first machine. Obviously, limitations would apply if one was to try this with the rootvg.

The import procedure is used to make the volume group known to a system after the group is exported and moved from another system. It is also used to reintroduce a volume group that was previously used on the system but had been exported. This ability is provided through the Volume Group Descriptor Area (VGDA) and the Logical Volume Control Block (LVCB).

Only one physical volume is required to identify the volume group; any remaining physical volumes (those belonging to the same volume group) are found by the `importvg` command (using the VGDA) and included in the import.

When a volume group with file systems is imported, the /etc/filesystems file is updated with values for the new logical volumes and mount points.

However, if the mount point information is longer than 128 characters, it will be missing from the logical volume control block, and then the /etc/filesystems file will have to be updated manually.

After import, the volume group can be activated with the `varyonvg` command (if not done by default). It is recommended at this time that the `fsck` be run against any file systems that were part of the volume group.

Notes

1. Importvg changed as part of AIX Version 4, such that the volume group is automatically varied on (activated) after it is imported unless the volume group is concurrent capable or imported with the `-c` flag. The `-n` flag will also stop this.
2. Volume groups can be shared between AIX Versions 3.2 to 4.3 so long as:

- Mirroring is not used with striping.
- A big VGDA is not used.
- The partition factor is not changed.
- Partition size of 512 or 1024 is not used.

The LVM is also designed to handle the accidental duplication of logical volume or file system names. If the LVM finds that a name already exists on the machine while the information is being imported, the LVM will create a new distinct name for the logical volume. It then prints a message and the new name to standard error and updates the /etc/filesystems file to include the new logical volume name. The following example shows how an imported logical volume was renamed by the LVM to avoid conflict.

We have two volume groups (t1vg and t2vg, which have their jfslog and one logical volume in common).

```
/home # lsvg -l t1vg
t1vg:
LV NAME            TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loglv00            jfslog     1     1     1    open/syncd    N/A
ulv01              jfs        1     1     1    open/syncd    /home/oscar
ulv02              jfs        4     4     1    open/syncd    /home/greg
/home #
/home # lsvg -l t2vg
t2vg:
LV NAME            TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loglv00            jfslog     1     1     1    open/syncd    N/A
ulv01              jfs        1     1     1    open/syncd    /home/oscar
ulv03              jfs        4     4     1    open/syncd    /home/sue
```

Volume group t1vg is active, and the file systems mounted.

```
/home # df
Filesystem    512-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4           16384      2296   86%     1281    32% /
/dev/hd2          884736     11888   99%    14863    14% /usr
/dev/hd9var       245760     81968   67%      170     1% /var
/dev/hd3           81920     44552   46%      109     2% /tmp
/dev/hd1           65536     39680   40%      296     4% /home
/dev/lv00       40009728  34429048   14%    10290     1% /software
/dev/lv05        1310720   1267248    4%       31     1% /logdata
/dev/ulv01         32768     31656    4%       18     1% /home/oscar
/dev/ulv02         32768     31656    4%       18     1% /home/greg
/home #
```

The second volume group is imported, and the following warning and error messages are received.

```
/home # importvg -yt2vg hdisk11
0516-530 synclvodm: Logical volume name ulv01 changed to fslv00.
0516-530 synclvodm: Logical volume name loglv00 changed to loglv02.
t2vg
imfs: Warning: logical volume fslv00 already exists in /etc/filesystems.
```

1.2.7.8 How importvg works

The logical volume manager provides another significant advantage to AIX: The ability to move a set of physical volumes among different AIX systems. By running `importvg`, the other AIX system learns:

1. Which physical volumes constitute the volume group.
2. What logical volumes exist on the physical volumes.
3. What mount points, if any, exist for the logical volumes.

The logical volume manager will prevent name duplication between the existing system and the information imported from the new physical volumes. Additionally, the logical volume manager creates the appropriate entries in /etc/filesystems and special files to allow any journaled file systems to be used. This is done by the LVM using the information that is contained in the VGDA, and it only needs the VGDA from one physical volume to find all the other physical volumes in the volume group. The key features of the VGDA in this regard are:

1. A list of the PVID's of all the physical volumes in the volume group.
2. A list of the LVIDs of all the logical volumes in the volume group.
3. A list of the preferred logical volume names associated with each of the LVIDs in the volume group.
4. Mount points for the jfs logical volumes in the volume group (in the big VGDA only).

Let us take a graphical view of an importvg of a volume group of two disks into a new AIX system. Here, we have two physical volumes, hdisk2 and hdisk3, that were part of a volume group on another system. We attach these two disks to a second RS/6000.

Before anything is done on the second system, it is aware that it has two new physical volumes but knows nothing about the contents. The `importvg` command is then used as follows:

```
importvg -y newvg hdisk2
```

Notice, that we only specified the first physical volume in the `importvg` command string. This is okay as we discussed above, the VGDA contains a list of what physical volumes make up the volume group. The LVM queries the VGDA on the first physical volume, checks VGDA consistency, and then builds a table of all the volume group and physical volume information. This data is then used to update the ODM with the key volume group and physical volume associations. The special files are created in /dev and /etc.

Next, the same action is preformed in regard to the logical volume information in the VGDA. All the logical volume information from the VGDA is read, the entries made in the ODM, and the special files created. The LVCB for each logical volume is also read. Then, each logical volume is checked to see if it has valid information relating to a journaled file system. If so, the appropriate stanzas are added to /etc/filesystems and the mount points created.

It is important to remember that, during this process, if the LVM finds any logical volumes defined in the VGDAs on the new physical volumes, they will be renamed so that they do not conflict with the existing system. These new names will be added back into the VGDA, LVCB, and used in the ODM of the system.

There is a common problem that occurs at this point if one of the logical volumes imported is a raw logical volume that has had its LVCB overwritten. This commonly occurs when the logical volume is used by a database application. In this instance, the LVM will not be able to fully populate the ODM with the logical volume information. Its logic will also stop the LVM from writing any information back to the LVCB, thus, avoiding any potential corruption of the data in the raw logical volume.

It is also worth noting, at this point, that even though the LVCB is corrupt, the logical volume is still stable and usable. If the logical volume is mirrored, this information will be kept in the VGDA, and mirroring through the LVDD will continue to operate. In previous releases of AIX, commands, such as `lslv`, query the ODM for their data and return incorrect data. For example, in the mirroring example, `lslv` will report copies as 1 (the default) even though the LVDD is correctly mirroring the raw logical volume. This has been corrected in the latest version of AIX, and `lslv` now correctly reflects the number of mirrors even if the LVCB has been overwritten.

1.2.7.9 Further options importing volume groups

Two new options have been added to the `importvg` command:

- Faster `importvg`
- Learning `importvg`

The faster import option gives the user the ability to speed up the importing of a volume group into a system. However, there are some risks in using this option and, thus, it is not the default option. The user must use the `-F` (fast import) option with care.

As discussed previously, the basis of all actions in LVM, when you are dealing with more than one disk in the volume group, is the volume group descriptor area. Each VGDA in a volume group is supposed to be identical, apart from the disk specific information. However, data integrity issues requires LVM to be particularly careful about the consistency of its data. So, whenever information is asked of the VGDA, a double-check is performed. LVM goes to each disk in the volume group and reads the starting and finishing timestamps in each VGDA. If any of the timestamps differ, then LVM knows something unusual has occurred the last time the volume group was accessed. The LVM uses the logic that the VGDA with the latest timestamp must be the correct one. It will then take this correct one and write over the inconsistent VGDAs. The LVM also checks every disk connected to make sure there are not any further disks with a matching volume group identifier. With a large number of disks attached to the system, this will obviously take time.

What the faster `importvg` does is, from the information in the VGDA of the disk supplied, assume that the all the VGDAs are consistent and proceed on the basis of that information.

Let us look at a specific example and show the difference in behavior between a normal and faster `importvg`.

Figure 8 shows the volume group, gromitvg, which consists of three physical volumes. As yet, hdisk4 has not been assigned.

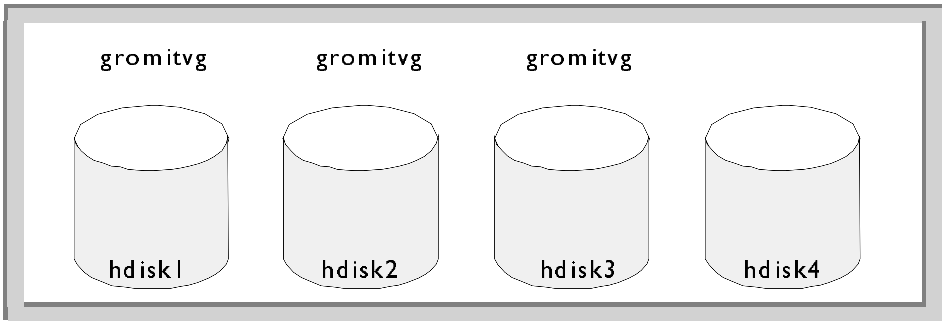


Figure 8. Three disks make up the volume group gromitvg

In Figure 9, hdisk1 is powered off. This is detected by the LVM, which updates the VGDA of hdisk2 and hdisk3.

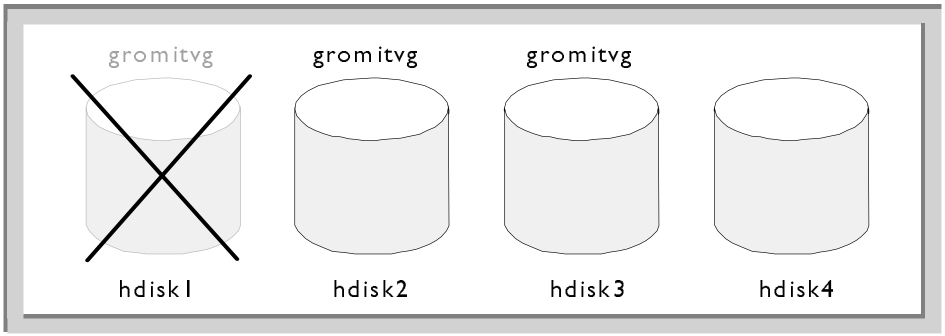


Figure 9. hdisk1 powered off and VGDA updated on disks 2 and 3

In Figure 10, hdisk4, is added to the volume group, and the VGDAs on hdisk2, hdisk3, and hdisk4 are updated.

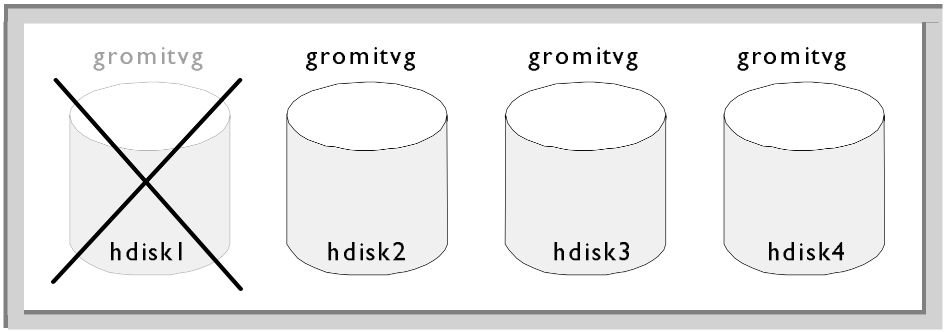


Figure 10. hdiks4 added to gromitvg

In Figure 11, we have the situation of all disks now available, but the VGDA on hdisk1 is out of date and doesn't know that hdisk4 is part of the volume group. The VGDAs on hdisk2, hdisk3, and hdisk4 are consistent and record that hdisk1 has stale information.

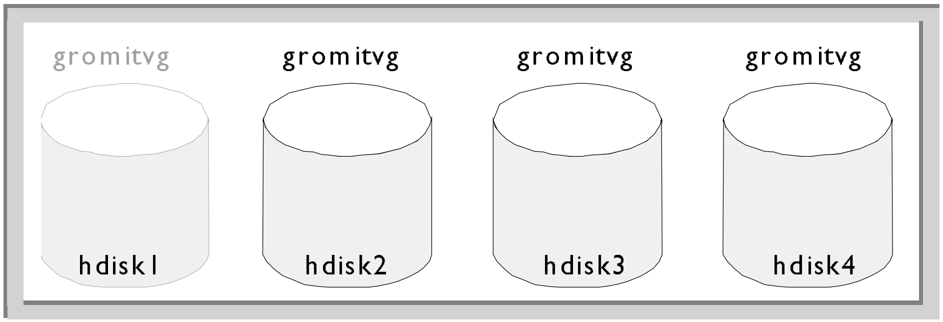


Figure 11. All disks now available, with inconsistent VGDAs

We now compare what happens with the normal and faster `importvg`.

Normal importvg

If the following command is run:

```
importvg -y gromitvg hdisk4
```

`importvg` will examine the VGDA on hdisk4 and get the VGID of `gromitvg`. It then activates every disk on the system to check for disks that match that VGID. As a result, `importvg` finds that hdisk1, hdisk2, hdisk3, and hdisk4 are members of `gromitvg`. The VGDA timestamps are then compared, and it is found that there is a problem with the hdisk1 VGDA. Two things are realized: hdisk1 is now alive, and its VGDA needs to be updated. This is done, and the VGDAs on all disks are then updated.

Using this logic, it can be seen that the same result would have been achieved if the `importvg` command was run against hdisk1. This is because all that `importvg` does, initially, is get the VGID, which it uses when it checks all attached disks.

Faster importvg

If we chose hdisk4 in the `importvg` command:

```
importvg -Fy gromitvg hdisk4
```

the same result as above would be achieved, as the VGDA on hdisk4 knows of the existence of all disks in `gromitvg`, and they will all be activated. All VGDAs will then be checked, the discrepancy found, and all updated.

However, if hdisk1 was chosen, then `importvg` will use the VGDA of hdisk1 to activate disks2 and hdisk3, leaving hdisk4 inactive. The VGDA discrepancy will be noticed and the VGDAs updated, but hdisk4 will not be made active and, therefore, will be labelled as missing.

This warning is well documented.

Note

> The `importvg -F` command provides a fast version of `importvg` that checks the Volume Group Descriptor Areas of only the disks that are members of the same volume group. As a result, if a user exercises this flag, they must ensure that all physical volumes in the volume group are in a good and known state. If this flag is used on a volume group where a disk may be in missing or removed state, the command may fail, or the results may be inconsistent.

This option will save time if there are a large number of physical volumes connected to the system, but the user has to be aware of the requirement for all disks to be available and in a good state.

1.2.7.10 The learning importvg

This command was created for multi-tailed systems mostly used in high-availability cluster multi-processing (HACMP) and Recoverable virtual shared disk (RVSD). The problem with twin- tailed systems is that a method is needed to communicate changes performed on one system to other systems that are connected. The concurrent LVM code takes care of this in a reasonable manner. However, the concurrent method depends on all nodes having the volume group varied on in concurrent mode. The learning option (`-L`) for `importvg` was designed to remove the need for all systems that access the shared physical volumes to export the volume groups and then re-import the new definitions. It is also used in conjunction with new options on the `varyonvg` command that allows the SCSI reserves (disk locks) that are set with a normal `varyonvg` to be broken. Details on this option can be found in the HACMP documentation.

1.2.7.11 Quorum

Quorum is the check used by the LVM to resolve possible data conflicts and to prevent data corruption. Quorum is a method by which 51 percent or more quorum votes must be available to a volume group before any LVM actions can continue. The only exception to this is when quorum is switched off (so that mirrored volume groups can continue to operate in highly-available environments) all VGDAs must be available and consistent when the volume group is made active. After that point, quorum is ignored.

The LVM runs most of its commands and strategies on the assumption of having the most current copy of the data. Thus, it needs a method to compare data on two or more physical volumes to determine which contains the most up to date information. If a quorum is lost during normal operation, and, therefore, the LVM will have trouble determining which copy is current, then the user will be notified that there will be no further I/O operations for that Volume Group, and the Volume Group is varied-off. In this case, you will see following entry on AIX error logging facility:

```
root@pukupuku:/ [274] # errpt -tj 91F9700D
Id       Label               Type CL Description
91F9700D LVM_SA_QUORCLOSE    UNKN H  QUORUM LOST, VOLUME GROUP CLOSING
root@pukupuku:/ [275] # errpt -taj 91F9700D
--------------------------------------------------------------------------
IDENTIFIER 91F9700D

Label: LVM_SA_QUORCLOSE
Class:     H
Type:      UNKN
Loggable: YES   Reportable: YES   Alertable: NO

Description
QUORUM LOST, VOLUME GROUP CLOSING

Probable Causes
PHYSICAL VOLUME UNAVAILABLE

Detail Data
MAJOR/MINOR DEVICE NUMBER
QUORUM COUNT
ACTIVE COUNT
```

The use of quorum does not guarantee that volume groups will be varied-off if a disk fails, as it is configured to work at the disk level, not the logical volume level. Imagine the simple case with four disks in the volume group, and one disk fails. The volume group still has quorum, but logical volumes may have lost a mirror copy or, if not mirrored, not be available at all.

In some high-availability configurations, a quorum buster disk may be used. This is an extra disk in the volume group that contains no information and no logical volumes. It is usually in a separate enclosure and allows quorum to be left on. With a quorum buster disk, more than 50 percent of the VGDAs will be available if one mirror copy is missing. Having quorum on allows a simpler recovery once the problem is fixed.

1.2.7.12 The process of varying on a volume group

The `varyonvg` and `varyoffvg` commands activate or deactivate (make available or unavailable for use) a volume group that you have defined to the system. The volume group must be varied on before the system can access it.

The vary-on process depends on whether quorum for the volume group is turned on or off:

On If quorum is on, then a majority of the VGDAs must be available before the process can complete. This process ensures that at least one copy of the VGDA is available and was online during the preceding vary off operation. This ensures the consistency of the VGDA.

Off All copies of the VDGA must be available before the volume group can be varied on. This can be over-ridden if the user forces the vary-on, or if the environment variable MISSINGPV_VARYON is set to `TRUE`. However, the user takes full responsibility for the volume group integrity.

If the vary-on operation cannot access one or more of the physical volumes defined in the volume group, the command displays the names of all physical volumes defined for that volume group and their status. This helps you decide whether to continue operating with this volume group.

It is during the vary-on process that the LVM reads the information in the VGDA/VGSA to initialize the logical volume device driver structures.

There is one exception to this quorum checking process. The `-f` flag can be used in the vary-on process, and this will allow the volume group to become active even if a majority of VGDAs are not available. All disks that cannot be put into an active state will be put in a removed state. At least one physical volume must be available. The environmental variable MISSINGPV_VARYON can also be set to TRUE. This must be set in /etc/environment to be effective on boot.

Note

Use of the force flag or the MISSINGPV_VARYON environment variable can result in data inconsistency. The user takes full responsibility for volume group integrity.

Let us look at a simple example involving two systems sharing two disks (one volume group with quorum off) that demonstrates the danger of forcing a vary-on of a volume group and, in fact, can lead to the loss of data.

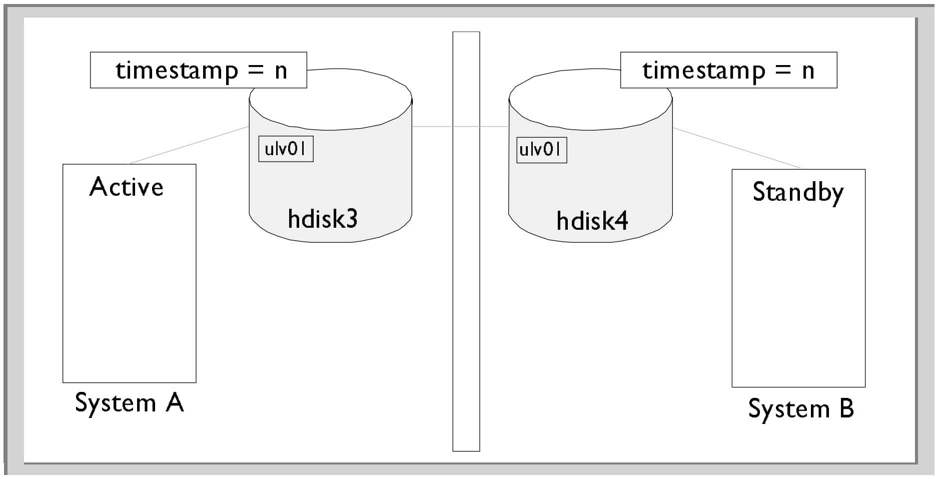


Figure 12. Two systems sharing two disks; system A is active.

At some time, due to a multiple failure, the second system becomes isolated. The first system recognizes this, and the VGDA of hdisk3 is updated with hdisk4 marked as missing. About this time, a new logical volume is created and will obviously only be on hdisk3 and the VGDA on hdisk3. The VGDA is updated with a new timestamp.

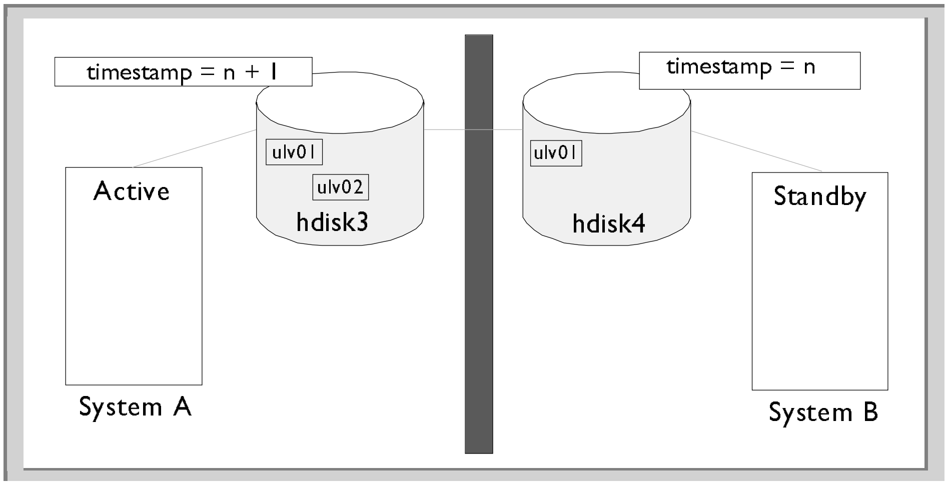


Figure 13. Physical problems cause system A and hdisk3 to become isolated

System B assumes that system A has problems; so, it attempts to activate the volume group containing hdisk3 and hdisk4. hdisk3 is found missing; so, a vary-on is forced, and the VGDA in hdisk4 is updated with hdisk3 missing and a later timestamp.

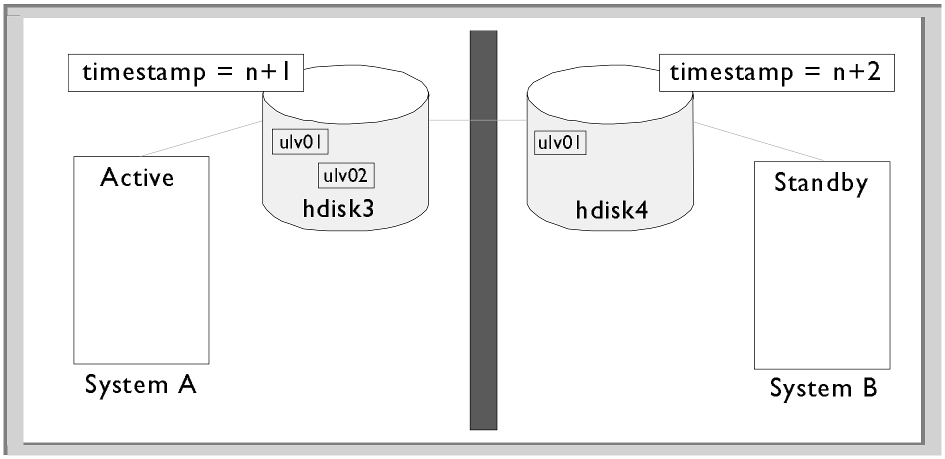


Figure 14. System B attempts to active the volume group

The problem is noted, both systems taken down and then brought up cleanly. Now System A sees the latest timestamp in the VGDA on disk4; so, any changes made after hdisk4 was isolated have been lost.

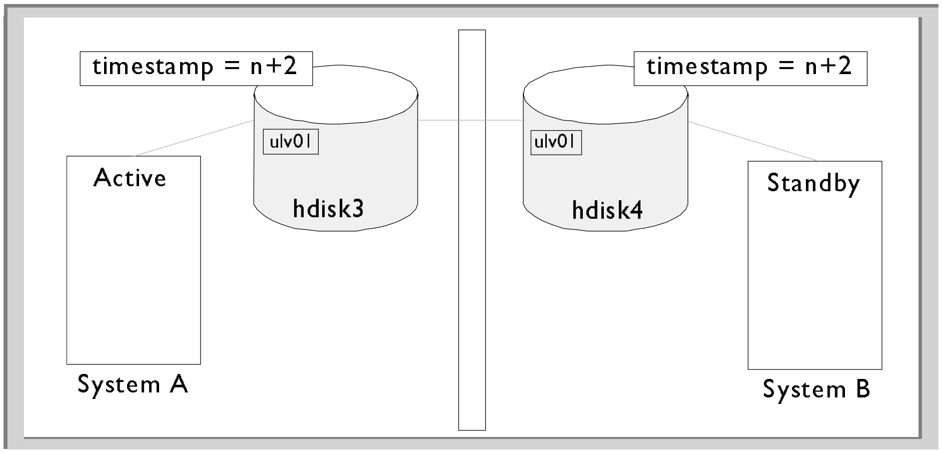


Figure 15. Systems stopped, then brought up cleanly - hdisk4's VGDA is used

Although this is an unlikely scenario, it demonstrates that one must understand the operation of the logical volume manager to ensure that the system will behave as expected if problems are encountered.

1.2.7.13 Querying the volume group

The `lsvg` command is useful in querying for configuration of the volume group.

The `lsvg vg_name` command will return high-level information about the volume group. This includes the VGID, the state, maximum number of physical partitions, physical partition size, and the physical partition usage.

```
/home/red # lsvg vikvg
VOLUME GROUP:   vikvg                    VG IDENTIFIER:  00017d377005016c
VG STATE:       active                   PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      535 (2140 megabytes)
MAX LVs:        512                      FREE PPs:       498 (1992 megabytes)
LVs:            3                        USED PPs:       37 (148 megabytes)
OPEN LVs:       0                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     1                        AUTO ON:        yes
MAX PPs per PV: 1016                     MAX PVs:        128
```

The `lsvg -p vg_name` command will return information about the physical volumes that make up the volume group.

```
/home/red # lsvg -p vikvg
vikvg:
PV_NAME           PV STATE    TOTAL PPs   FREE PPs    FREE DISTRIBUTION
hdisk11           active      535         498         107..70..107..107..107
```

The `lsvg -l vg_name` command will return information about the logical volumes in the volume group.

```
/home/red # lsvg -l vikvg
vikvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
ulv01               jfs        12    12    1    closed/syncd  /home/a_dent
ulv02               jfs        24    24    1    closed/syncd  /home/f_prefect
loglv00             jfslog     1     1     1    closed/syncd  N/A
```

1.2.8 Logical volumes

A logical volume is a portion of a physical volume or volumes viewed by the system as a single unit. Logical records are records defined in terms of the information they contain rather than physical attributes. Logical volumes consist of logical partitions, each of which will map to one or more physical partitions.

The logical volume presents a simple contiguous view of data storage to the application/user while hiding the more complex and possibly non-contiguous physical placement of data.

Logical volumes can only exist within one volume group. They cannot be mirrored nor expanded into other volume groups. You will recall from the discussion earlier in this chapter, that the logical volume information is kept in the VGDA, and the VGDA only tracks information pertaining to one volume group.

The creation of logical volumes allows many levels of control by the user:

- No control
- Physical volume specific control
- Partition location control

The command that determines how the physical partitions are allocated to the user's request is `/usr/sbin/allocp`. This binary takes into account the user's specific requirements and default values and is called by `mklv` (see commands). It is also this command that uses map files to lay out the logical volume onto specific physical partitions.

The Logical Volume Manager subsystem provides flexible access and control for complex physical storage systems. Logical volumes can be mirrored and stripped with strip sizes of 4 K, 8 K, 16 K, 32 K, 64 K and 128 K. Mirroring and stripping was not supported for AIX Version 4.3.2 and below.

There are a number of logical volumes that exist on the system that are addressed in an atypical manner:

- Log logical volume - Used by jfs.
- Dump logical volume - Used by the system dump facility to copy selected areas of kernel data when an unexpected system halt occurs.
- Boot logical volume - Contains the initial information required to start the system.
- Paging logical volumes - Used by the virtual memory manager to swap out pages of memory. From AIX 4.1 on, the paging and dump devices can be the same.

Users and applications will only use the following types of logical volume:

Raw Logical Volumes These will be controlled by the application (usually a database application) and will not use the journaled file system (JFS).

Journaled file systems Each JFS consists of a pool of page-sized (4 KB) blocks. When data is to be written to a file, one or more additional blocks are allocated to that file. These blocks may or may not be contiguous with one and another or with other blocks previously allocated to the file. In AIX 4.1 and above, a given file system can be defined as having a fragment size of less than 4 KB (512 bytes, 1 KB, 2 KB).

1.2.8.1 The logical volume identifier

The logical volume identifier (LVID) is the soft serial number that consists of the VGID of the volume group, a decimal point, and the minor number for the logical volume special file. This number also represents the order in which the logical volume was created within the volume group.

1.2.8.2 The logical volume control block

The logical volume control block (LVCB)

- Occupies the first 512 bytes of a logical volume

- Can be damaged by direct access to the logical volume.
- LVCB is not required for many LVM operations.
- With the introduction of the big VGDA, a copy of the LVCB is now kept in the VGDA.

The LVCB holds important information, such as the creation date of the logical volume, information about mirrored copies and mount points, if the logical volume is a journaled file system. The command, `/usr/sbin/getlvcb`, can be used to interrogate the LVCB.

A number of LVM commands are required to update the LVCB, and to ensure consistency within the LVM, the old LVCB is read first and analyzed to confirm its validity. If the information is confirmed, the LVCB is updated. If, however, there is a problem, the LVCB is not updated, and a warning message is issued:

```
Warning, cannot write lv control block data
```

This most commonly occurs if the user has created a raw logical volume for direct access by an application. Often, in this instance, the application writes directly over the LVCB. However, this is not fatal, and the user can still perform commands on the logical volume, but some LVM commands require a valid LVCB to complete successfully.

1.2.8.3 Logical partitions

Each logical volume consists of a sequence of numbered partitions. Logical partitions are how logical volumes handle both the mapping to physical partitions on one or more physical volumes and multiple copies for mirroring. As noted before, the logical partitions appear contiguous to the user/application, while the physical partitions may well not be. Mirroring is supported by each logical partition being mapped to more than one physical partition. Figure 2 on page 2 shows this relationship.

The location of the physical partitions is determined by the allocation policies.

1.2.8.4 Control over location of logical volumes

The following terms will need to be understood before we can go on with what AIX does when creating logical volumes:

Inter physical volume allocation policy

This determines the position of the physical partitions on the physical volume that will be used by the logical volume. The options are outer edge, outer middle, center, inner middle, and inner edge (Figure 3 on page 3). SMIT uses slightly different terminology (Table 5). Once this region is full, further partitions are allocated as near as possible to the first region (details in Table 6.). The first free physical partitions on the selected region will be allocated first, ensuring that no holes are left in that region.

Intra-physical volume allocation policy

This determines the number (or range) of physical volumes that the physical partitions will be allocated across. The physical volumes that provide the best match for the inter-physical volume policy will be used. The number of physical volumes can be limited by the upper bound variable. This can be set to minimum or maximum.

Strict allocation policy

This determines whether copies of a logical partition share the same physical volume or not. There are three options :

Strict — Copies of a logical partition cannot share the same physical volume.

Not strict — Copies of a logical partition can share the same physical volume.

Super strict — Physical partitions from one copy of a logical partition cannot share the same physical volume as physical partitions from another copy of other logical partitions of the same logical volume.

Scheduling — This determines how physical partitions will be written when there is more than one copy. The options are parallel (written at the same time) or sequential (written one after another).

Mirror write consistency — Mirror write consistency ensures data consistency among mirrored copies of a logical volume during normal I/O processing. If a system or volume group is not shutdown properly, then mwc will identify which logical partitions may be inconsistent.

Write verify — This controls the write verify state for the logical volume. There are two options:

y — All writes to the logical volume will be verified by a following read.

n Writes are not verified.

Map files Map files layout the logical partition to physical volume and physical partition mapping explicitly. By using map files, the exact location of the logical volume can be specified.

Bad block relocation This determines if bad block relocation is to take place. There are two options: y or n.

There are some discrepancies in the disk's regions naming conventions between SMIT and the AIX documentation, Table 5 gives you the mapping between the two groups of description.

Table 5. SMIT region terms for intra-physical allocation policy

SMIT	Description
edge	outer edge
middle	outer middle
center	center
inner middle	inner middle
inner edge	inner edge

Table 6 gives you the regions that will be allocated when you specify one region but that region doesn't have enough free partitions to fulfill your request.

Table 6. Order physical volume regions are searched for free partitions.

Inter policy region selected	Other regions allocated in order
outer edge	outer middle, center, inner middle, inner edge
outer middle	center, outer edge, inner middle, inner edge
center	outer middle, inner middle, outer edge, inner edge
inner middle	center, inner edge, outer middle, outer edge
inner edge	inner middle, center, outer middle, outer edge

While the performance and availability aspects of logical volume placement will be discussed in detail in later chapters, it is important to understand the default behavior of logical volume creation. Therefore, we will look at differing levels of control that can be introduced.

Preference rules

The following table (Table 7) shows the priority that `mklv`/`extendlv` commands will use to allocate physical partitions depending on the different options selected.

Table 7. Order in which physical partitions are allocated

Allocatn Rule	Disk	Contiguous PPs pool	Region	Closest to selected	Min Intra policy
mklv options:					
default		1	2	3	4
disk	1	2	3	4	5
region			1	2	3
extendlv options:					
default		1	3	2	4
disk	1	2	3		
region		2	1	4	3

Thus, if `mklv` is used, and region is selected, then if a contiguous pool of free physical partitions cannot be found in the specified region on any of the disks, as many physical partitions will be allocated from that region on one disk followed by free partitions in neighboring regions. In this case, the minimum policy wins out.

Note

In versions of AIX before 4.2, allocp would chose contiguous partitions over region specificity.

Default behavior

The default options shown in Table 8 will be used if a logical volume is created with no user control.

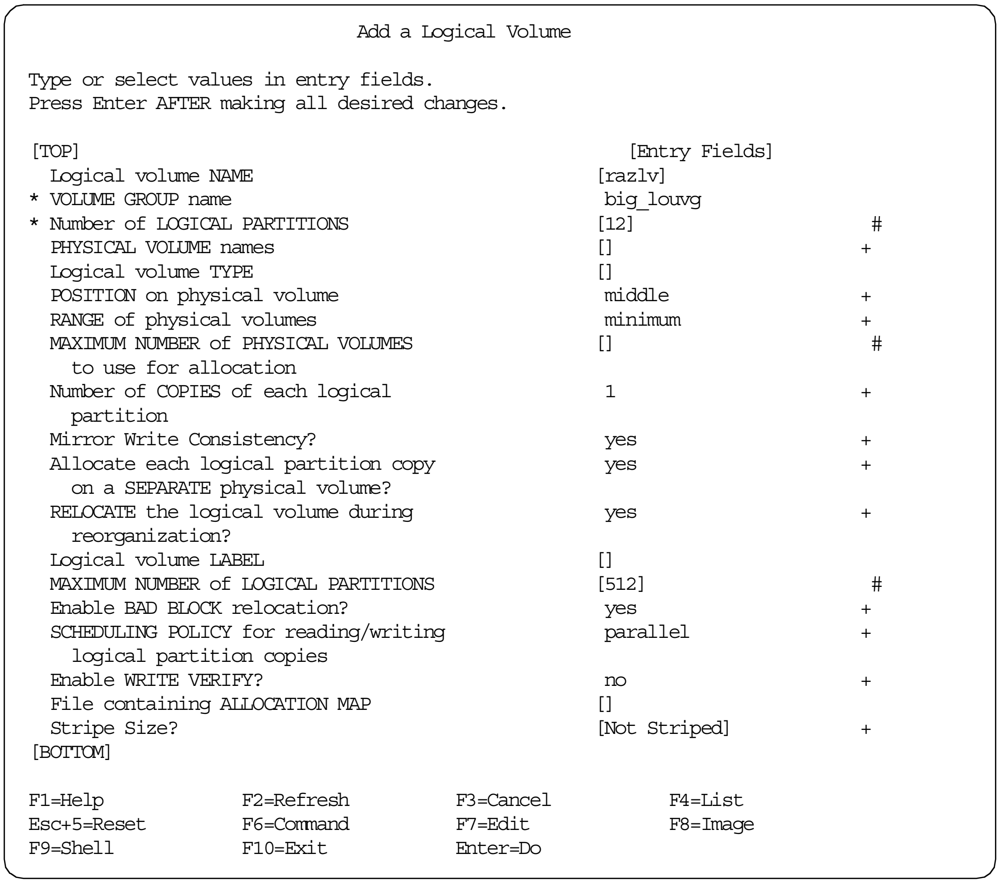
Table 8. Default configuration of a logical volume

Option	Value
Logical volume type	jfs
Inter pv allocation policy	The physical partitions in the outer middle (SMIT middle) will be allocated first.

Option	Value
Intra pv allocation policy	The logical volume will be created across the minimum number of physical volumes possible.
Max number of PV (Upper bound)	Number of physical volumes / number of copies
Number of copies	1
Mirror write consistency	yes
strictness	yes
bad block relocation	yes
scheduling policy	parallel
write verify	no

The following screen shows the SMIT screen with the default options.

Default options for creating a logical volume.



```
                         Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                  [Entry Fields]
  Logical volume NAME                               [razlv]
* VOLUME GROUP name                                  big_louvg
* Number of LOGICAL PARTITIONS                      [12]                  #
  PHYSICAL VOLUME names                             []                   +
  Logical volume TYPE                               []
  POSITION on physical volume                        middle              +
  RANGE of physical volumes                          minimum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                []                    #
    to use for allocation
  Number of COPIES of each logical                   1                   +
    partition
  Mirror Write Consistency?                          yes                 +
  Allocate each logical partition copy               yes                 +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                 yes                 +
    reorganization?
  Logical volume LABEL                              []
  MAXIMUM NUMBER of LOGICAL PARTITIONS              [512]                 #
  Enable BAD BLOCK relocation?                       yes                 +
  SCHEDULING POLICY for reading/writing              parallel            +
    logical partition copies
  Enable WRITE VERIFY?                               no                  +
  File containing ALLOCATION MAP                    []
  Stripe Size?                                      [Not Striped]        +
[BOTTOM]

F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

A slightly greater level of control over the placement of the logical volume can be exercised by using the inter- and intra-physical volume allocation policies. The greatest level of control can be achieved through the use of a map file.

1.2.8.5 The logical volume map file

A map file is used to specify exactly which physical partitions on which physical volumes will contain the logical partitions for each copy of a logical volume. The physical partitions are allocated to logical partitions for a logical volume copy according to the order in which they appear in the map file. Each logical volume should have its own map file, and the map files of each logical volume copy should each allocate the same number of physical partitions. Hence, it offers very precise control when a logical volume is first created and when copies are subsequently created in a mirrored environment.

We can see this in an example of the current usage of a physical volume:

lspv -p showing usage of physical volume by region.

```
hdisk9:
PP RANGE  STATE   REGION        LV ID             TYPE       MOUNT POINT
  1-108   free    outer edge
109-128   used    outer middle  tlv               jfs        N/A
129-215   free    outer middle
216-322   free    center
323-429   free    inner middle
430-537   free    inner edge
```

Or, the detailed use of each physical partition:

lslv -p hdisk9 showing physical partition usage.

```
hdisk9:::
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE      1-10
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     11-20
                    ............................
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     91-100
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE                  101-108

USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    109-118
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    119-128
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    129-138
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    139-148
USED   USED   USED   USED   USED   FREE   FREE   FREE   FREE   FREE    149-158
                    ............................
```

The map file specifies the physical volume and the physical partition (or range of partitions):

Example of a map file, which allows exact placement.

```
hdisk9:90
hdisk9:93
hdisk9:96-150
hdisk9:230
hdisk9:235
hdisk9:240
hdisk9:244
hdisk9:250
hdisk9:257
hdisk9:160-164
hdisk9:170
hdisk9:175
```

Note

When using a map file to specify which physical partitions are to be used, you can ignore the inter-disk allocation and intra-physical volume allocation policies in SMIT as the map file will take precidence.

If we are creating a mirrored logical volume, then the physical partitions on each physical volume must be specified. For example, if we are creating a logical volume with two copies, we would use a map file as follows:

Example map file for a logical volume with two copies.

```
/home/red # cat mapf
hdisk9:90
hdisk9:93
hdisk9:96
hdisk9:230
hdisk9:235
hdisk10:90
hdisk10:93
hdisk10:96
hdisk10:230
hdisk10:235
```

Then, create the logical volume:

Creating a mirrored logical volume using a map file

```
/home/red # mklv -t jfs -y roslv -c 2 -m ./mapf shazvg 5
roslv
```

with the resulting logical to physical partition mapping:

The resulting logical volume:

```
/home/red # lslv -m roslv
roslv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0090 hdisk9            0090 hdisk10
0002  0093 hdisk9            0093 hdisk10
0003  0096 hdisk9            0096 hdisk10
0004  0230 hdisk9            0230 hdisk10
0005  0235 hdisk9            0235 hdisk10
```

Note

Map files cannot be used to create striped logical volumes.

1.2.8.6 Reorganizing logical volumes

It is important that the placement of logical volumes and the `reorgvg` command is understood and not confused with file system fragmentation. File system fragmentation affects i-nodes and data blocks within a file system and will covered in Chapter 5, “The AIX journaled file system” on page 233.

The reorgvg command works only at the logical volume level and rearranges the physical partitions within the volume group to do its best to conform with the placement policy for that logical volume.

At least one free physical partition must exist in the specified volume group for reorgvg to be able to run.

The following example shows how the physical partitions are rearranged for one of the logical volumes. Two logical volumes are created (suelv and douglv) from map files and are scattered over the physical volume.

One logical volume (suelv) has the relocate flag set to yes.

```
/home/red # lslv suelv
LOGICAL VOLUME:     suelv                  VOLUME GROUP:   keovg
LV IDENTIFIER:      00017d376c4734b6.1     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        4 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                13                     PPs:            13
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    128
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV ?: yes
```

One logical volume (douglv) has the relocate flag set to no.

```
home/red # lslv douglv
LOGICAL VOLUME:     douglv                 VOLUME GROUP:   keovg
LV IDENTIFIER:      00017d376c4734b6.2     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        4 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                10                     PPs:            10
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    no
INTRA-POLICY:       middle                 UPPER BOUND:    128
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV ?: yes
```

Here, is the layout of the physical partitions for logical volume suelv:

```
/home/red # lslv -m suelv
suelv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0090 hdisk11
0002  0093 hdisk11
0003  0096 hdisk11
0004  0230 hdisk11
0005  0235 hdisk11
0006  0240 hdisk11
0007  0244 hdisk11
0008  0250 hdisk11
0009  0257 hdisk11
0010  0160 hdisk11
0011  0164 hdisk11
0012  0170 hdisk11
0013  0175 hdisk11
/home/red #
```

Here, is the layout of the physical partitions for logical volume douglv:

```
/home/red # lslv -m douglv
douglv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0100 hdisk11
0002  0104 hdisk11
0003  0106 hdisk11
0004  0110 hdisk11
0005  0113 hdisk11
0006  0118 hdisk11
0007  0094 hdisk11
0008  0097 hdisk11
0009  0128 hdisk11
0010  0124 hdisk11
```

The volume group is reorganized. Note, in AIX Version 4.2 and above, partitions belonging to every logical volume in the volume group specified will be rearranged if no other options are given.

```
/home/red # reorgvg keovg
0516-962 reorgvg: Logical volume suelv migrated.
/home/red #
```

The mapping between the logical partitions and the physical partitions is changed for logical volume suelv.

```
/home/red # lslv -m suelv
suelv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0129 hdisk11
0002  0130 hdisk11
0003  0131 hdisk11
0004  0132 hdisk11
0005  0133 hdisk11
0006  0134 hdisk11
0007  0135 hdisk11
0008  0136 hdisk11
0009  0137 hdisk11
0010  0138 hdisk11
0011  0139 hdisk11
0012  0140 hdisk11
0013  0141 hdisk11
/home/red #
```

The physical partitions for logical volume douglv are not changed.

```
/home/red # lslv -m douglv
douglv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0100 hdisk11
0002  0104 hdisk11
0003  0106 hdisk11
0004  0110 hdisk11
0005  0113 hdisk11
0006  0118 hdisk11
0007  0094 hdisk11
0008  0097 hdisk11
0009  0128 hdisk11
0010  0124 hdisk11
/home/red #
```

1.2.8.7 Understanding logical volumes and bad blocks

When a system fails to perform a read or a write due to physical problems with the physical volume, data relocation typically occurs. With some physical volumes, data relocation will be handled internally, and the user will not be notified. In other cases, depending on the error returned, the logical volume device driver may chose to relocate the data, as it is uncertain of the success of future I/O to that portion of the physical volume.

When a bad block is detected during I/O, the physical disk driver sets an error condition to indicate that there was a media surface error. The physical layer of the LVDD then initiates any bad block processing that must be done.

If the operation was a non-mirrored read, then the block is not relocated immediately, as the data in the relocated block is not initialized until a write is performed to that block. To support this delayed action, an entry for the bad block is put in the LVDD defects directory and into the bad block directory on the disk. These entries contain no relocation block address, and the status is set to indicate that relocation is required. On subsequent I/O requests, the physical layer checks to see if there are any bad blocks in the request. If there is a write to a block in the relocation required state, then the relocation is done at this point.

If the operation was a mirrored request, a request to read one of the other mirrors is initiated. If this second read was successful, then the read is turned into a write request, and the relocation attempted. The user is not notified in this instance, but the physical device driver does log the permanent I/O errors.

The LVDD can handle bad block relocation in the following ways:

Hardware relocation directed by the LVDD

In this instance, the LVDD will instruct the physical volume to relocate the data on one physical partition to the reserved part of the physical volume. This reserved part of the disk is a pool of relocation blocks at the end of the physical volume. The physical device driver copies the data to the next available relocation block. The LVDD, however, continues to refer to the data at its original physical location and relies on the physical volume to handle the I/O to the new location.

This form of relocation is sometimes known as safe hardware relocation and is not allowed on non-IBM drives. It is only with IBM drives that the LVDD is guaranteed a reserved area for possible relocation (typically 256 blocks per disk)

Software relocation

In this instance, the LVDD device driver maintains a bad block directory, and it will relocate blocks in the relocation pool at the end of the disk. So, whenever the LVDD receives a request to access a specific logical location, it looks up its own bad block table to find the actual physical location.

With all I/O requests, the physical layer of the logical volume device driver checks to see if there are any known software related bad blocks in the request.

If a physical volume needs to be formatted, including reinitializing any bad blocks, the format function supplied by the `diag` command should be used.

> **Note**
>
> Bad block relocation only takes place for disks smaller than 128 GB.

1.2.8.8 Querying the logical volume

The `lslv` command is useful in querying a configuration of the logical volume.

The `lslv lv_name` command will return high-level information about the logical volume. This includes the LVID, the volume group, allocation policies, and status.

```
/home/red # lslv ulv02
LOGICAL VOLUME:     ulv02                  VOLUME GROUP:   keovg
LV IDENTIFIER:      00017d377005016c.2     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        4 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                24                     PPs:            24
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    128
MOUNT POINT:        /home/kim              LABEL:          /home/kim
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV ?: yes
```

The `lslv -l lv_name` command will show the distribution of the logical volume’s physical partitions across the five regions (in order from outside in).

```
/home/red # lslv -l ulv02
ulv02:/home/kim
PV                COPIES        IN BAND       DISTRIBUTION
hdisk11           024:000:000   100%          000:024:000:000:000
```

The `lslv -m lv_name` command will give the detailed map file for the logical volume.

```
/home/red # lslv -m ulv02
ulv02:/home/kim
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0120 hdisk11
0002  0121 hdisk11
0003  0122 hdisk11
0004  0123 hdisk11
0005  0124 hdisk11
0006  0125 hdisk11
0007  0126 hdisk11
0008  0127 hdisk11
0009  0128 hdisk11
0010  0129 hdisk11
0011  0130 hdisk11
0012  0131 hdisk11
0013  0132 hdisk11
0014  0133 hdisk11
0015  0134 hdisk11
0016  0135 hdisk11
0017  0136 hdisk11
0018  0137 hdisk11
0019  0138 hdisk11
0020  0139 hdisk11
0021  0140 hdisk11
0022  0141 hdisk11
0023  0142 hdisk11
0024  0143 hdisk11
```

The `lslv -m hdiskx` command will show the status of each physical partition.

```
/home/red # lslv -p hdisk11
hdisk11:::
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE      1-10
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     11-20
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     21-30
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     31-40
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     41-50
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     51-60
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     61-70
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     71-80
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     81-90
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE     91-100
FREE   FREE   FREE   FREE   FREE   FREE   FREE                          101-107

USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    108-117
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    118-127
USED   USED   USED   USED   USED   USED   USED   USED   USED   USED    128-137
USED   USED   USED   USED   USED   USED   USED   FREE   FREE   FREE    138-147
                     .................................
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    509-518
FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE   FREE    519-528
FREE   FREE   FREE   FREE   FREE   FREE   FREE                          529-535
```

1.2.8.9 Non typical logical volumes

So far, we have seen raw logical volumes and logical volumes that contain a journaled file system. There are a number of other common types of logical volume types:

paging This is a logical volume that is used by the system for paging memory.

jfslog Used as the log for a journaled file system.

boot Used to hold the boot image.

copy Used as the destination logical volume in a "logical volume copy" process.

sysdump Used by the kernel as the dump space.

1.2.9 Mirroring

AIX supports two types of mirrors:

Sequential There is a distinct primary copy and secondary copy. Writes to a sequential logical volume are done to each copy at a time, waiting on the completion of each write before moving onto the next. Reads are done using the primary copy.

Parallel There is no distinct primary and copy; all copies in a mirrored set can just be referred to as a copy. This is the most common type in AIX. As there is no ordering of the copies, a copy can be removed or added at any time. Writes are done in parallel, and reads will use the disks with the least number of outstanding I/Os. This will be a balance between location of the read/write head and other activity on the physical volume.

If a copy of a physical partition in a mirrored logical volume becomes unavailable for any reason, the logical volume device driver notifies the VGSA that a particular physical partition on that physical volume is stale. This will halt any further reads or writes being issued to that particular physical volume, and once the physical volume becomes available again, the re-synchronization code will use this information to bring all copies of the mirror up-to-date.

1.2.10 Striping

Striping is the technique for spreading the data in a logical volume across several physical volumes in such a way that the I/O capacity of the physical volumes can be used in parallel to access the data. This functionality is not offered before AIX Version 4, and striping and mirroring were only introduced in AIX Version 4.3.3. The primary objective of striping is very high performance I/O for large sequential files.

In an ordinary logical volume, the data addresses correspond to the sequence of blocks in the underlying physical partitions. In a striped logical volume, the data addresses follow the sequence of strip units. A complete strip unit consists of one stripe on each of the physical volumes that contain part of the striped logical volume. The LVM determines which physical block on which physical volume corresponds to the block being read or written. If more than one physical volume is involved, the necessary I/O operations are scheduled simultaneously.

Note

While mirrored copies can be added and removed from existing logical volumes, an existing logical volume cannot be converted to a striped logical volume.

Further information is contained in Chapter 3, “Striping” on page 137.

1.2.11 The ODM

When volume groups are created or imported, the information that is contained in the VGDA and LVCB is also stored on the system. There are some files in /etc that will contain information used by the LVM, such as /etc/filesystems. Other information is kept in the object data manager (ODM).

Similarly, information about physical volumes is also kept in the ODM

1.2.11.1 How LVM Works with ODM

The main LVM commands, such as `lsvg`, `lspv`, and `lslv` work with both the ODM and the VGDA to gather their information. For older versions of AIX, these commands don’t work directly with the VGDA but work with what are commonly called the vg special files. These special files are located in /etc/vg and are distinguished by the vg< VGID> naming method.

In the original 3.1 version of LVM, the idea was that a file copy of the VGDA would be kept in the /etc/ vg/vg* files. The commands used to describe LVM would then consist of information gathered from ODM and the /etc/ vg files. This was thought as a quick way of accessing data without disk I/O to the system. However, two main problems occurred during the use of this design in 3.1 and 3.2. The first problem was that every so often, the information held in /etc/vg/vg* would be out of sync with what was truly held in the VGDA. The second, more common problem, was when /etc would fill up during the creation of a new volume group. Every new non-rootvg volume group caused a file of roughly 1 MB to be created. However, sometimes the /etc directory would be filled up before the entire /etc/vg special file was completely written. This caused an ambiguous state within the volume group. It could be accessed, but certain commands would fail because the vg special file was “chopped off”. The only solution to this problem was to `exportvg` the volume, expand the root directory, and then remake the volume group. But, by the time this limitation was realized as the full root directory problem of some other LVM problems, the user already had data sitting in the volume group.

In AIX 4.1, the solution to these two problems was to have the logical volume manager commands reference both the ODM and the VGDA, with the /etc/vg/vg* file used purely for locking/synchronization purposes during configuration. So, the `lsvg`, `lspv`, and `lslv` commands now go to the actual VGDA on the platter to gather their information. This design frees up /etc space and guarantees the user the absolute latest data about the volume group and its attributes.

Another aspect of LVM’s involvement with the ODM is the ODM object known as the vg- lock. Whenever an LVM modification command is started, the LVM command will go to the vg’s entry in ODM and put a vg- lock. This tells other LVM processes that may want to access and change the same volume group. However, in some cases, the vg- lock gets left behind, and jobs are inadvertently turned away. There are two solutions to this: The user can unlock the vg with the simple command:

```
putlvodm -K <VGID>
```

(this vg was locked in the first place with `putlvodm -k <VGID>`)

Or, the user can simply `varyonvg` the volume group. The `varyonvg` command always clears any vg locks, which might explain why leftover locks are never seen after a system reboot. And to many people’s surprise, `varyonvg` can be run on a volume group that is already varied on. No harm done here as long as the volume group is not part of some twin- tailed configuration (then you’d have to take locking conditions from the `varyon` into account).

1.2.11.2 Physical Volume stanzas

The ODM contains information about each physical volume:

- Adapter attributes
- SCSI/SSA attributes
- Size (in MB)

- Physical volume ID
- Major and minor numbers

The following is an example of CuAt stanzas with physical volume information:

```
CuAt:
name = "hdisk11"
attribute = "size_in_mb"
value = "2255"
type = "R"
generic = "D"
rep = "nr"
nls_index = 60

CuAt:
name = "hdisk11"
attribute = "pvid"
value = "00017d37e671fe4b0000000000000000"
type = "R"
generic = "D"
rep = "s"
nls_index = 15
```

The following is an example of CuAt stanzas with physical volume adapter information:

```
CuAt:
name = "hdisk11"
attribute = "connwhere_shad"
value = "0006294E6EE600D"
type = "R"
generic = "D"
rep = "s"
nls_index = 55

CuAt:
name = "hdisk11"
attribute = "adapter_a"
value = "ssa0"
type = "R"
generic = "D"
rep = "s"
nls_index = 51

CuAt:
name = "hdisk11"
attribute = "primary_adapter"
value = "adapter_a"
type = "R"
generic = "DU"
rep = "sl"
nls_index = 52

CuAt:
name = "hdisk11"
attribute = "queue_depth"
value = "3"
type = "R"
generic = "DU"
rep = "nr"
nls_index = 1

CuAt:
name = "hdisk11"
attribute = "max_coalesce"
value = "0x20000"
type = "R"
generic = "DU"
rep = "nr"
nls_index = 57
```

The following is an example of CuDep stanza with physical volume information:

```
CuDep:
name = "hdisk11"
dependency = "ssa0"
```

The following is an example of CuDvDr stanzas with physical volume information:

```
CuDv:
name = "hdisk11"
status = 1
chgstatus = 2
ddins = "ssadisk"
location = "30-60-L"
parent = "ssar"
connwhere = "0006294E6EE600D"
PdDvLn = "disk/ssar/hdisk"

CuDvDr:
resource = "devno"
value1 = "30"
value2 = "8199"
value3 = "hdisk11"
```

1.2.11.3 Volume group stanzas

The ODM contains information about each volume group:

- Volume group ID
- Each physical volume ID
- Size (in MB)
- Volume group timestamp
- Each logical volume
- Status
- Major and minor numbers

The following is an example of CuAt stanzas with volume group information:

```
CuAt:
name = "park_vg"
attribute = "pv"
value = "00017d37e671fe4b0000000000000000"
type = "R"
generic = ""
rep = "sl"
nls_index = 0

CuAt:
name = "park_vg"
attribute = "vgserial_id"
value = "00017d378edf7c9d"
type = "R"
generic = "D"
rep = "n"
nls_index = 637

CuAt:
name = "park_vg"
attribute = "timestamp"
value = "37f375a70e77ee20"
type = "R"
generic = "DU"
rep = "s"
nls_index = 0
```

The following is an example of CuDep stanzas with volume group information:

```
CuDep:
name = "park_vg"
dependency = "grom_lv"

CuDep:
name = "park_vg"
dependency = "wall_lv"

CuDep:
name = "park_vg"
dependency = "loglv00"
```

CuDv stanzas with volume group information

```
CuDv:
name = "park_vg"
status = 1
chgstatus = 1
ddins = ""
location = ""
parent = ""
connwhere = ""
PdDvLn = "logical_volume/vgsubclass/vgtype"

CuDv:
name = "loglv00"
status = 1
chgstatus = 1
ddins = ""
location = ""
parent = "park_vg"
connwhere = ""
PdDvLn = "logical_volume/lvsubclass/lvtype"

CuDv:
name = "grom_lv"
status = 1
chgstatus = 1
ddins = ""
location = ""
parent = "park_vg"
connwhere = ""
PdDvLn = "logical_volume/lvsubclass/lvtype"

CuDv:
name = "wall_lv"
status = 1
chgstatus = 1
ddins = ""
location = ""
parent = "park_vg"
connwhere = ""
PdDvLn = "logical_volume/lvsubclass/lvtype"
```

The following is an example of CuDvDr stanzas with volume group information:

```
CuDvDr:
resource = "ddins"
value1 = "park_vg"
value2 = "42"
value3 = ""

CuDvDr:
resource = "devno"
value1 = "42"
value2 = "0"
value3 = "park_vg"
```

1.2.11.4 Logical volume stanzas

The ODM contains information about each logical volume:

- Logical volume ID
- Stripe width
- Size in physical partitions
- Label
- Status
- Major and minor numbers

The following is an example of a CuAt stanzas with logical volume information:

```
CCuAt:
name = "grom_lv"
attribute = "lvserial_id"
value = "00017d378edf7c9d.1"
type = "R"
generic = "D"
rep = "n"
nls_index = 648

CuAt:
name = "grom_lv"
attribute = "stripe_width"
value = "0"
type = "R"
generic = "DU"
rep = "r"
nls_index = 1100

CuAt:
name = "grom_lv"
attribute = "label"
value = "/home/gromit"
type = "R"
generic = "DU"
rep = "s"
nls_index = 640

CuAt:
name = "grom_lv"
attribute = "size"
value = "12"
type = "R"
generic = "DU"
rep = "r"
nls_index = 647
```

The following is an example of a CuDep stanza with logical volume information:

```
CuDep:
name = "park_vg"
dependency = "grom_lv"
```

The following is an example of a CuDv stanza with logical volume information:

```
CuDv:
name = "grom_lv"
status = 1
chgstatus = 1
ddins = ""
location = ""
parent = "park_vg"
connwhere = ""
PdDvLn = "logical_volume/lvsubclass/lvtype"
```

The following is an example of a CuDvDr stanza with logical volume information:

```
CuDvDr:
resource = "devno"
value1 = "42"
value2 = "1"
value3 = "grom_lv"
```

1.3 Operation of the logical volume manager

The previous section gave an overview of the structure of the logical volume manager. Now, we will look at the individual components. 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 16 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.

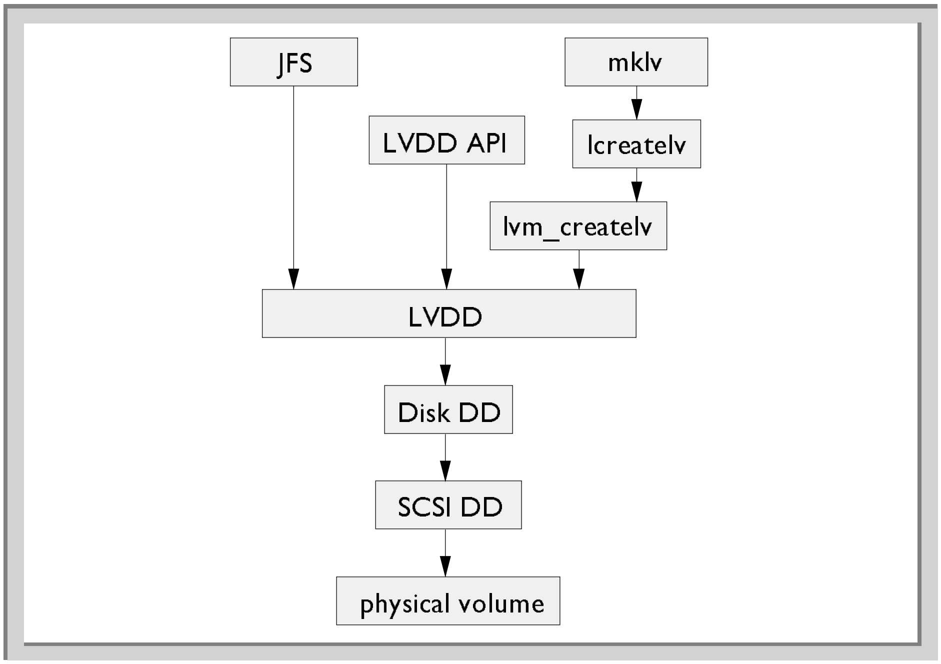


Figure 16. Flow from high level commands to the physical volume

1.3.1 High-level commands

These are commands that are:

- Used by SMIT
- Used directly by experienced users
- Have a high-level of error checking

Most of the LVM commands reside in /usr/sbin (except `/etc/cfgvg`), and they are a combination of shell scripts and C binaries:

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

* indicates that the command is a binary

1.3.2 Intermediate-level commands

These are commands that are:

- Used by high-level commands

- Provide usage statements
- Some level of error checking provided, but documentation on output

The intermediate commands found in /usr/sbin are designed to be called by the high-level commands and are all C binaries:

```
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.
```

Note

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.

1.3.3 Logical Volume Manager subroutine interface library

The Logical Volume Manager subroutine interface library contains routines that are used by the system management commands to perform system management tasks for the logical and physical volumes of a system.

The programming interface for the library is available to anyone who wishes to expand the function of the system management commands for logical volumes.

- Library calls can be used as Application Programme Interface.
- Error checking provided.
- Calls documented in the AIX documentation and books.
- Very rarely used.

The LVM library calls are part of the LVM Applications Programmer Interface (API) that let programmers access the LVM layer directly through the library layer provided by /usr/lib/liblvm.a. In our experience, very few people use this API because of the complexity of issues within LVM. Users mostly rely the high-level commands. The API library functions begin with lvm_ and are documented within Infoexplorer. These commands are only called in situations where the user is writing C code and does not want to access the high-level shell scripts from within a C program. Or, they require a faster response from LVM than they can achieve with shell scripts. Writing an LVM specific program using the API certainly does give a performance improvement to some applications. Note, the LVM library API is not a thread-safe library. Thus, user programmes written for threads that reference the LVM API may not work correctly.

The library functions reside in /usr/lib/liblvm.a and /usr/lib/libsm.a.

lvm_changelv This subroutine changes the attributes of an existing logical volume.

lvm_changepv This subroutine changes the attributes of a physical volume in a volume group.

lvm_createlv This subroutine creates an empty logical volume in a specified volume group.

lvm_createvg This subroutine creates a new volume group and installs the first physical volume. The physical volume cannot be part of another volume group.

lvm_deletelv This subroutine deletes the logical volume from its volume group. The logical volume must not be open, and the volume group must be varied on. Also, all logical partitions belonging to the logical volume must be removed before it can be deleted.

lvm_deletepv This subroutine deletes a physical volume from its volume group. The logical volume must be varied on and on logical volumes on the physical volume.

lvm_extendlv This subroutine extends a logical volume by the specified number of partitions. This is done by adding a completely new logical partition or by adding a copy to an existing logical partition.

lvm_installpv This subroutine installs a physical volume into a volume group. The physical volume must not exist in another volume group.

lvm_migratepp This subroutine moves the specified physical partition from the first physical volume to the specified physical partition on the second physical volume within the one volume group. The volume group must be varied on.

lvm_querylv This subroutine queries a logical volume and returns the current state of the logical volume, the current size in logical partitions, the physical partition size, the permission assigned, bad block relocation flag, write verify flag, mirror write consistency flag, open or closed flag, and an array of pointers to the partition map lists for each copy of the logical partitions.

lvm_querypv This subroutine queries a physical volume and returns the physical partition size, current state, total number of physical partitions, number of physical partitions allocated, the number of VGDAs, and an arrary of information about each physical partition.

lvm_queryvg This subroutine queries a volume group and returns the maximum number of logical volumes, physical partition size, free physical partitions, number of logical volumes, number of physical volumes, number of VGDAs for the volume group, arrary of LVIDs, names and states and an array of the PVIDs, and it states and number of VGDAs for each physical volume

lvm_queryvgs This subroutine queries volume groups and returns the number of online volume groups and an array of their VGIDs and major numbers.

lvm_reducelv This subroutine reduces the size of a logical volume by the specified number of partitions. The logical volume should be closed and the volume group online. On partial reductions, all the remaining partitions must have at least one active (non-stale) copy, and the last active partition cannot be reduced.

lvm_resynclp This subroutine synchronizes all physical partitions for a logical partition.
The force parameter can be used:

False A good copy of each partition is propagated to only the stale copies of the logical partition.

True A good copy of each partitions is propagated to all other copies of the logical partition. This is sometimes necessary in cases where mirror write consistency recovery was not specified for the logical volume lvm_resynclv This subroutine synchronizes all physical copies of all the logical partitions for a logical volume. The force parameter (above) can be used.

lvm_resyncpv This subroutine synchronizes all physical partitions on a physical volume with the related copies of the logical partition to which they correspond.The volume group must be varied on at the time. The force parameter (above) can be used.

lvm_varyoffvg This subroutine varies off a volume group. All logical volumes in the volume group must be closed.

lvm_varyonvg This subroutine varies on a volume group. The physical volumes in the volume group are examined and the vgda restored if necessary.

Further information on these routines can be found in the *AIX Version 4.3 Technical Reference: Base Operating System and Extensions , Volume 1*, SC23-4160.

1.3.4 Introduction to device drivers

Whenever an operating system component or an application needs to interact with a physical device, such as a disk or tape, it does so through services provided by another element of the operating system known as a device driver. Device drivers provide a common mechanism for accessing the devices that they support. Device drivers are treated as though they are ordinary files; thus, when a process needs to communicate with a device, it uses the open subroutine to initialize the interface and can then use the read and write calls to access data. Control of the device is accomplished using the ioctl calls, and when the task requiring the device is complete, the interface is ended with the close subroutine.

There are two main types of device driver: Those designed for character oriented devices, such as terminals, and printers, and those designed for block devices, such as disks. Storage management devices are usually block devices, as this is more efficient for transfer of larger quantities of information.

Device drivers normally consist of two parts:

The top half This half is responsible for interacting with requestors, blocking and buffering data, queuing up requests for service, and handling error recovery and logging.

The bottom half This half is responsible for the actual I/O to and from the device and can perform some preprocessing on requests, such as sorting reads from disks for maximum efficiency.

Device drivers provide the primary interface to devices. More information can be found in the *AIX Version 4.3 Technical Reference: Kernel and Subsystems*, SC23-4163.

1.3.5 Logical volume device driver

The Logical Volume Manager consists of the Logical Volume Device Driver (LVDD) and the Logical Volume Manager subroutine interface library. The logical volume device driver is a pseudo-device driver that manages and processes all I/O and operates on logical volumes through the /dev/lvn 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.

The logical volume device driver comes in two portions: The pinned portion and the non pinned portion. The driver's true non-pageable portion is in hd_pin_bot while the hd_pin portion is pageable. Prior to AIX 4.1, the driver was just called hd_pin and the entire driver was pinned into memory (that is, was not pageable).

The logical volume device driver is either called by the jfs file system or lvm library routines. When a request is received by the LVDD, it calls the physical disk device driver.

Character I/O requests are performed by issuing a read or write request on a /dev/rlvn character special file for a logical volume. The read or write is processed by a file system supervisor call (SVC) handler that calls the LVDD ddread or ddwrite entry point. The ddread or ddwrite entry point transforms the character request into a block request. This is done by building a buffer for the request and calling the LVDD ddstrategy entry point.

Block I/O requests are performed by issuing a read or write on a block special file /dev/lvn for a logical volume. These requests go through the supervisor call handler to the bread or bwrite block I/O kernel services. These services build buffers for the request and call the LVDD ddstrategy entry point. The

LVDD ddstrategy entry point then translates the logical address to a physical address (handling bad block relocation and mirroring) and calls the appropriate physical disk device driver.

On completion of the I/O, the physical device driver calls the iodone kernel service on the device interrupt level. This service then calls the LVDD I/O completion handling routine. Once this is completed, the LVDD calls the iodone service again to notify the requester that the I/O is completed.

The LVDD is logically split into top and bottom halves. The top half contains the ddopen, ddclose, ddread, ddwrite, ddioctl, and ddconfig entry points. The top half runs in the context of a process address space and can page fault. It contains the following entry points as shown in Table 9.

Table 9. LVDD entry points

Entry Point	Description
ddopen	Called by the file system, when a logical volume is mounted, to open the logical volume specified.
ddclose	Called by the file system, when a logical volume is unmounted, to close the logical volume specified.
ddconfig	Initializes data structures for the LVDD.
ddread	Called by the read subroutine to translate character I/O requests to block I/O requests. This entry point verifies that the request is on a 512-byte boundary and is a multiple of 512 bytes in length. When a character request spans partitions or logical tracks (32 pages of 4 K bytes each), the LVDD ddread routine breaks it into multiple requests. The routine then builds a buffer for each request and passes it to the LVDD ddstrategy entry point, which handles logical block I/O requests. If the ext parameter is set (called by the readx subroutine), and the character device is opened. The ddread entry point passes this parameter to the LVDD ddstrategy routine in the b_options field of the buffer header.
ddwrite	Called by the write subroutine to translate character I/O requests to block I/O requests. The LVDD ddwrite routine performs the same processing for a write request as the LVDD ddread routine does for read requests.

Entry Point	Description
ddioctl	Supports the following operations: CACLNUP Causes the mirror write consistency (MWC) cache to be written to all physical volumes (PVs) in a volume group. IOCINFO, XLATE Return LVM configuration information and PP status information. LV_INFO Provides information about a logical volume. This ioctl operation is available beginning with AIX Version 4.2.1 PBUFCNT Increases the number of physical buffer headers (pbufs) in the LVM pbuf pool.

The bottom half of the LVDD contains the ddstrategy entry point, which contains block read and write code. This is done to isolate the code that must run fully pinned and has no access to user process context. The bottom half of the device driver runs on interrupt levels and is not permitted to page fault.

In particular, the bottom half of the LVDD:

- Validates I/O requests. Checks requests for conflicts (such as overlapping block ranges) with requests currently in progress.
- Translates logical addresses to physical addresses.
- Handles mirroring and bad-block relocation.

The bottom half of the LVDD runs on interrupt levels and, as a result, is not permitted to page fault. The bottom half of the LVDD is divided into the following three layers:

- Strategy Layer
- Scheduler Layer
- Physical Layer

Each logical I/O request passes down through the bottom three layers before reaching the physical disk device driver. Once the I/O is complete, the request returns back up through the layers to handle the I/O completion processing at each layer. Finally, control returns to the original requestor.

1.3.5.1 Strategy layer

The strategy layer deals only with logical requests. The ddstrategy entry point is called with one or more logical buf structures. A list of buf structures for requests that are not blocked are passed to the second layer, the scheduler.

1.3.5.2 Scheduler layer

The scheduler layer schedules physical requests for logical operations and handles mirroring and the MWC cache. For each logical request the scheduler layer schedules one or more physical requests. These requests involve translating logical addresses to physical addresses, handling mirroring, and calling the LVDD physical layer with a list of physical requests.

When a physical I/O operation is complete for one phase or mirror of a logical request, the scheduler initiates the next phase (if there is one). If no more I/O operations are required for the request, the scheduler calls the strategy termination routine. This routine notifies the originator that the request has been completed.

The scheduler also handles the MWC cache for the volume group. If a logical volume is using mirror write consistency, then requests for this logical volume are held within the scheduling layer until the MWC cache blocks can be updated on the target physical volumes. When the MWC cache blocks have been updated, the request proceeds with the physical data write operations.

When MWC is being used, system performance can be adversely affected. This is caused by the overhead of logging or journaling that a write request does when active in a logical track group (LTG) (32 4K-byte pages or 128K bytes). This overhead is for mirrored writes only. It is necessary to guarantee data consistency between mirrors, particularly if the system crashes before the write to all mirrors has been completed.

Mirror write consistency can be turned off for an entire logical volume. It can also be inhibited on a request basis by turning on the NO_MWC flag as defined in the /usr/include/sys/lvdd.h file.

1.3.5.3 The Physical Layer

The physical layer of the LVDD handles startup and termination of the physical request. The physical layer calls a physical disk device driver's ddstrategy entry point with a list of buf structures linked together. In turn, the physical layer is called by the iodone kernel service when each physical request is completed.

This layer also performs bad-block relocation and detection/correction of bad blocks, when necessary. If the physical volume is capable, then the LVDD will make use of the hardware function, otherwise, the LVDD will handle it totally. These details are hidden from the other two layers.

The only variation to this configuration would be if HAGEO/GeoRM (the IBM products that support geographically remote mirroring) or VSD (Virtual Shared Disk) are installed.

With the HAGEO/GeoRM products, a new logical device, the GeoMirror Device (GMD), is added between the application and the Logical Volume Device Driver. This pseudo-device has a local and remote comment and controls the replication of data between sites.

The VSD software is used to make AIX logical volumes appear to be located on more than one node in an SP Cluster. However, only the logical volume can be accessed using the VSD software, not the mounted file systems. Further information can be found in the product documentation *SP Administration Guide*, GC23-3897.

1.3.6 Disk device driver

There are various disk device drivers in AIX. 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 SSA disks.

1.3.6.1 SCSI disk device drivers

SCSI disk device drivers support the small computer system interface (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.

The scdisk SCSI device driver uses raw and block special files in performing its functions.

Table 10 lists the special files used by the scdisk device driver.

Table 10. SCSI device driver special files (in /dev)

	Character (raw) access	**Block access**
Fixed disks	rhdisk0, rhdisk1,...rhdiskn	hdisk0, hdisk1, ... hdiskn
Read write CD	romd0, romd1, ... romdn	omd0, omd1, ... omdn

1.3.6.2 SSA disk device drivers

Serial Storage Architecture (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.

By default:

- One pdisk is always configured for each physical disk drive.
- One hdisk is configured for each disk drive that is connected to the active system. If the disks are part of an array, then one hdisk is configured for each array.

SSA physical disks have the following properties:

- Are configured as pdisk0, pdisk1.....pdiskn.
- Have errors logged against them in the system error log.
- Support a character special file (/dev/pdisk0, /dev/pdisk1..../dev/pdiskn).
- Support the ioctl subroutine for servicing and diagnostics functions.
- Do not accept read or write subroutine calls for the character special file.

SSA logical disks have the following properties:

- Are configured as hdisk0, hdisk1.....hdiskn.
- Support a character special file (/dev/rhdisk0, /dev/rhdisk1..../dev/rhdiskn).
- Support a block special file (/dev/hdisk0, /dev/hdisk1..../dev/hdiskn).
- Support the ioctl subroutine call for non service and diagnostics functions only.
- Accept the read and write subroutine call to the special files.
- Can be members of volume groups and have file systems mounted upon them.

Multiple Adapters

Some SSA subsystems allow a disk drive to be controlled by up to two adapters in a particular system. The disk drive has, therefore, two paths to each using system, and the SSA subsystem can continue to function if an adapter fails. If an adapter fails, or the disk drive becomes inaccessible from the original adapter, the SSA disk device driver switches to the alternative adapter without returning an error to any working application.

Once a disk drive has been successfully opened, the takeover by the alternative adapter does not occur because the drive becomes reserved or fenced out. However, during an open of a ssa logical disk, the device driver does attempt to access the disk drive through the alternative adapter if the path through the original adapter experiences reservation conflict or fenced-out status.

Configuring SSA disk drive devices.

(SSA) disk drives are represented in AIX as SSA Logical disks (hdisk0, hdisk1.....hdiskN) and SSA physical disks (pdisk0,pdisk1.....pdiskn). Normally, all the disk drives connected to the system will be configured automatically by the system boot process, and the user will need to take no action to configure them.

Since some SSA devices may be connected to the SSA network while the system is running without taking the system off-line, it may be necessary to configure SSA disks after the boot process has completed. In this case, the devices should be configured by running the configuration manager with the `cfgmgr` command.

Device Attributes

SSA logical disks, SSA physical disks, and the SSA router have several attributes.You can use the `lsattr` command to display these attributes. Some of the attributes are

- Names of the connected adapters
- Name of the primary adapter
- pvid
- Queue depth
- max_coalesce
- Reserve lock

Table 11. SSA device driver special files (in /dev)

	Character (raw) access	**Block access**
logical SSA drives	rhdisk0, rhdisk1,...rhdiskn	hdisk0, hdisk1, ... hdiskn
physical SSA drives	pdisk1, pdisk2, ... pdiskn	

1.3.7 Adapter device driver

1.3.7.1 SCSI adapter device driver

There are several SCSI device drivers for AIX, and they have the common name string of "scsi" residing as part of their name. The most common of them is the original SCSI-1 device driver /usr/lib/drivers/hscsidd. The SCSI device driver takes a command from a SCSI device, such as tape, disk, CD-ROM, or scanner, and processes it to be sent to the SCSI device connected onto the SCSI bus. The SCSI device is device neutral. It does not know or care which device it sends the command to or even what the command is. It treats all requests the same and puts them into the same type of SCSI command packaging required by all SCSI devices.

1.3.7.2 SSA adapter device driver

The SSA adapter device driver provides an interface that allows application programmes to access SSA adapters and SSA devices that are connected through these adapters. The SSA adapter has a number of ODM attributes that can be displayed using the `lsattr` command, for example, adapter microcode file name, interrupt levels, and so on.

1.4 Use of the logical volume manager

To achieve the most from the logical volume manager, you will need to properly plan the configuration of your volume groups, your logical volumes, and really understand what the logical volume manager offers.

You will also need to be aware of restrictions that are imposed by the logical volume manager. These will be dealt with in the next section.

1.4.1 Planning your use of volume groups

Disk failure is the most common hardware failure in the storage system, followed by failure of adapters and power supplies. Protection against disk failure primarily involves the configuration of the logical volumes. However, volume group size also plays a part, as will be explained.

To protect against adapter and power supply failure, you need to consider a special hardware configuration for any specific volume group. Such a configuration includes two adapters and at least one physical volume per adapter with mirroring across adapters and a non-quorum volume group configuration. The additional expense for this kind of configuration is not appropriate for all sites or systems. It is recommended only where high (up-to-the-last-second) availability is a priority. Depending on the configuration, high availability can cover hardware failures that occur between the most recent backup and the current data entry. High availability does not apply to files deleted by accident.

1.4.1.1 When to Create Separate Volume Groups

You should organize physical volumes into volume groups separate from rootvg for the following reasons:

- For safer and easier maintenance.
  - Operating system updates, reinstallations, and crash recoveries are safer because you can separate user file systems from the operating system so that user files are not jeopardized during these operations.
  - Maintenance is easier because you can update or reinstall the operating system without having to restore user data. For example, before updating, you can remove a user-defined volume group from the system by un-mounting its file systems and deactivating it (using `varyoffvg`). If necessary, the volume could even be exported (using `exportvg`). After updating the system software, you can reintroduce the user-defined volume group (using `importvg`), active it (`varyonvg`), then remount its file systems.
- For different physical-partition sizes. All physical volumes within the same volume group must have the same physical partition size. To have physical volumes with different physical partition sizes, place each size in a separate volume group.
- When different quorum characteristics are required. If you have a file system for which you want to create a non-quorum volume group, maintain a separate volume group for that data; all of the other file systems should remain in volume groups operating under a quorum.
- For security. For example, you might want to de-active a volume group at night
- To switch physical volumes between systems. If you create a separate volume group for each system on an adapter that is accessible from more than one system, you can switch the physical volumes between the systems that are accessible on that adapter without interrupting the normal operation of either (see the `varyoffvg`, `exportvg`, `importvg`, and `varyonvg` commands).
- To remove physical volumes from the system while the system continues to run normally. By making a separate volume group for removable physical volumes, provided the volume group is not rootvg, you can make removable physical volumes unavailable and physically remove them during normal operation without affecting other volume groups.

1.4.1.2 High availability with physical volume failure

The primary methods used to protect against physical volume failure involve logical volume configuration settings, such as mirroring. While the volume group considerations are secondary, they have significant economic implications because they involve the number of physical volumes per volume group:

- The quorum configuration, which is the default, keeps the volume group active (varied on) as long as a quorum (51 percent) of the physical volumes is present. In most cases, you need at least three physical volumes with mirrored copies in the volume group to protect against physical volume failure.
- The non-quorum configuration keeps the volume group active (varied on) as long as one VGDA is available on a physical volume (see "Changing a Volume Group to Non-quorum Status"). With this configuration, you need only two physical volumes with mirrored copies in the volume group to protect against physical volume failure.

When deciding on the number of physical volumes in each volume group, you also need to plan for room to mirror the data. Keep in mind that you can only mirror and move data between physical volumes that are in the same volume group. If the site uses large file systems, finding disk space on which to mirror could become a problem at a later time. Be aware of the implications on availability of inter-physical volume settings for logical volume copies and intra-physical volume allocation for a logical volume.

1.4.1.3 High availability in case of adapter or power supply failure

To protect against adapter or power supply failure, depending on the stringency of your requirements, do one or more of the following:

- Use two adapters, located in the same or different cabinets. Locating the adapters in different cabinets protects against losing both adapters if there is a power supply failure in one cabinet.
- Use two adapters, attaching at least one physical volume to each adapter. This protects against a failure at either adapter (or power supply if adapters are in separate cabinets) by still maintaining a quorum in the volume group, assuming cross-mirroring (copies for a logical partition cannot share the same physical volume) between the logical volumes on physical volume A (adapter A) and the logical volumes on physical volume B (adapter B). This means that you copy the logical volumes that reside on the physical volumes attached to adapter A to the physical volumes that reside on adapter B, and also that you copy the logical volumes that reside

on the physical volumes attached to adapter B to the physical volumes that reside on adapter A as well.

- Configure all physical volumes from both adapters into the same volume group. This ensures that at least one logical volume copy will remain intact in case an adapter fails, or if cabinets are separate, in case a power supply fails.
- Make the volume group a non-quorum volume group. This allows the volume group to remain active as long as one Volume Group Descriptor Area (VGDA) is accessible on any physical volume in the volume group (see "Changing a Volume Group to Non-quorum Status").
- If there are two physical volumes in the volume group, implement cross-mirroring between the adapters. If more than one physical volume is available on each adapter, implement double-mirroring. In this case, you create a mirrored copy on a physical volume that uses the same adapter and one on a physical volume using a different adapter.

1.4.1.4 Decide on the size of physical partitions

The physical partition size is set when the volume group is created. The default size is 4 MB. The default is designed to suit most sites and systems but may not be appropriate in every case. You can choose a partition size as small as 1 MB to gain flexibility in sizing, but this requires more partitions. The additional partitions create more overhead for the logical volume manager and are likely to affect performance.

If you make the partitions larger than 4 MB, you lose some sizing flexibility and may also waste space. For example, if you have 32 MB partitions, then your JFS log will have to be 32 MB when it only needs 4 MB. Some waste may be an acceptable trade-off if the particular site or system requires larger partitions.

Note

You may only create and extend physical partitions in increments that are a factor of their size; for example, 32 MB partitions are created or extended in 32 MB increments.

1.4.2 Planning your use of logical volumes

The policies described in this section help you set a strategy for logical volume use that is oriented toward a combination of availability, performance, and cost that is appropriate for your site.

Availability is the ability to recover data that is lost because of physical volume, adapter, or other hardware problems. The recovery is made from copies of the data that are made and maintained on separate physical volumes and adapters during normal system operation.

Performance is the average speed at which data is accessed. Policies, such as write-verify and mirroring, enhance availability but add to the system processing load and, thus, degrade performance. Mirroring doubles or triples the size of the logical volume. In general, increasing availability degrades performance. Physical volume striping can increase performance, but mirroring and striping are only supported at AIX Version 4.3.3 and above.

By controlling the allocation of data on the physical volume and between physical volumes, you can tune the storage system for the highest possible performance. See Chapter 7, “Performance” on page 303, or the *AIX Performance Tuning Guide*, SC23-2365, for detailed information on how to maximize storage-system performance.

The sections that follow should help you evaluate the trade-offs between performance, availability, and cost. Remember that increased availability often decreases performance, and vice versa. Mirroring may increase performance; however, the LVM chooses the copy on the least busy physical volume for reads.

Note

Mirroring does not protect against the loss of individual files that are accidentally deleted or lost because of software problems. Mirroring is not a substitute for conventional backup arrangements.

This section discusses:

- Determining the scheduling policy for mirrored writes
- Determining mirror write consistency policy
- Choosing an inter-physical volume allocation policy
- Choosing an intra-physical volume allocation policy
- Combining allocation policies
- Using map files for precise allocation
- Determining a write-verify policy
- Implementing volume group policies

The following issues will also need to be kept in mind:

1. Determine whether the data that will be stored in the logical volume is valuable enough to warrant the processing and disk-space costs of mirroring.
2. Performance and mirroring are not always opposed. If the different instances (copies) of the logical partitions are on different physical volumes, preferably attached to different adapters, the LVM can ensure that read performance degrades less rapidly by reading the copy on the least busy disk. Writes, unless physical volumes are attached to different adapters, always cost the same because you must update all copies.
3. If you have a large sequential-access file system that is performance-sensitive, you should consider physical volume striping.
4. Normally, whenever data on a logical partition is updated, all the physical partitions containing that logical partition are automatically updated. However, physical partitions can become stale (no longer containing the most current data) because of system malfunctions or because the physical volume was unavailable at the time of an update. The LVM can refresh stale partitions to a consistent state by copying the current data from an up-to-date physical partition to the stale partition. This process is called mirror synchronization. The refresh can take place when the system is restarted, when the physical volume comes back online, or when the `syncvg` command is issued.
5. Any change that affects the physical partition makeup of a boot logical volume requires that you run `bosboot` after that change. This means that actions, such as changing the mirroring of a boot logical volume, require a `bosboot`.
6. A dump to a mirrored logical volume results in an inconsistent dump and therefore, should be avoided. Since the default dump device is the primary paging logical volume, you should create a separate dump logical volume if you mirror your paging logical volumes; and, therefore, if you mirror your root volume group, you should create a separate dump logical volume also.

1.4.2.1 Determine scheduling policy for mirrored writes

For data that has only one physical copy, the logical volume device driver (LVDD) translates a logical read or write request address into a physical address and calls the appropriate physical device driver to service the request. This single-copy or non-mirrored policy handles bad block relocation for write requests and returns all read errors to the calling process.

If you use mirrored logical volumes, two different scheduling policies for writing to disk can be set for a logical volume with multiple copies, sequential and parallel.

The sequential-scheduling policy performs writes to multiple copies or mirrors in order. The multiple physical partitions representing the mirrored copies of a single logical partition are designated primary, secondary, and tertiary. In sequential scheduling, the physical partitions are written to in sequence; the system waits for the write operation for one physical partition to complete before starting the write operation for the next one.

The parallel-scheduling policy starts the write operation for all the physical partitions in a logical partition at the same time. When the write operation to the physical partition that takes the longest to complete finishes, the write operation is completed.

For read operations on mirrored logical volumes with a sequential-scheduling policy, the primary copy is read. If that read operation is unsuccessful, the next copy is read. During the read retry operation on the next copy, the failed primary copy is corrected by the LVM with a hardware relocation. Thus, the bad block that prevented the first read from completing is patched for future access.

Specifying mirrored logical volumes with a parallel-scheduling policy may improve I/O read-operation performance because multiple copies allow the system to direct the read operation to the disk with the least number of outstanding I/Os.

1.4.2.2 Determine mirror write consistency policy

Mirror Write Consistency (MWC) identifies which logical partitions may be inconsistent if the system or the volume group is not shut down properly. When the volume group is varied back online for use, this information is used to make logical partitions consistent again.

If a logical volume is using MWC, then requests for this logical volume are held within the scheduling layer until the MWC cache blocks can be updated on the target physical volumes. When the MWC cache blocks have been updated, the request proceeds with the physical data write operations.

When MWC is being used, system performance can be adversely affected. This is caused by the overhead of logging or journaling that a write request is active in a Logical Track Group (LTG) (32 4 K-byte pages or 128 K bytes). This overhead is for mirrored writes only. It is necessary to guarantee data consistency between mirrors only if the system or volume group crashes before the write to all mirrors has been completed. When MWC is not used, the mirrors of a mirrored logical volume can be left in an inconsistent state in the event of a system or volume group crash.

After a crash, any mirrored logical volume that has MWC turned off should do a forced sync (`syncvg -f -l LVname`) before the data within the logical volume is used. With MWC turned off, writes outstanding at the time of the crash can leave mirrors with inconsistent data the next time the volume group is varied on. An exception to this is logical volumes whose content is only valid while the logical volume is open, such as paging spaces.

A mirrored logical volume is really no different to a non-mirrored logical volume with respect to a write. When LVM completely finishes with a write request, the data has been written to the drive(s) below LVM. Until LVM issues an iodone on a write, the outcome of the write is unknown. Any blocks being written that have not been completed (iodone) when a machine crashes should be rewritten whether mirrored or not and regardless of the MWC setting.

MWC only makes mirrors consistent, when the volume group is varied back online after a crash, by picking one mirror and propagating that data to the other mirrors. MWC does not keep track of the latest data; it only keeps track of LTGs currently being written; therefore, MWC does not guarantee that the latest data will be propagated to all the mirrors. It is the application above LVM that has to determine the validity of the data after a crash. From the LVM prospective, if the application always reissues all outstanding writes from the time of the crash, the possibly inconsistent mirrors will be consistent when these writes finish (as long as the same blocks are written after the crash as were outstanding at the time of the crash).

1.4.2.3 Choose an inter-physical volume allocation policy

The inter-physical volume allocation policy specifies the number of physical volumes on which a logical volume's physical partitions are located. The physical partitions for a logical volume might be located on a single physical volume or spread across all the physical volumes in a volume group. Two options in the `mklv` and `chlv` commands are used to determine inter-physical volume policy:

- The `Range` option determines the number of physical volumes used for a single physical copy of the logical volume.
- The `Strict` option determines whether the `mklv` operation will succeed if two or more copies must occupy the same physical volume.

- Striped logical volumes can only have a maximum range and a strict or super strict inter-physical volume policy.

Inter-physical settings for a single copy of the logical volume
If you select the minimum inter-physical volume setting (range = minimum), the physical partitions assigned to the logical volume are located on a single physical volume to enhance availability. If you select the maximum inter-physical volume setting (Range = maximum), the physical partitions are located on multiple physical volumes to enhance performance. The allocation of mirrored copies of the original partitions is discussed in the following section.

For non-mirrored logical volumes, use the minimum setting to provide the greatest availability (access to data in case of hardware failure). The minimum setting indicates that one physical volume should contain all the original physical partitions of this logical volume, if possible. If the allocation program must use two or more physical volumes, it uses the minimum number while remaining consistent with other parameters.

By using the minimum number of physical volumes, you reduce the risk of losing data because of a disk failure. Each additional physical volume used for a single physical copy increases that risk. An non-mirrored logical volume spread across four physical volumes is four times as likely to lose data because of one physical volume failure than a logical volume contained on one physical volume.

Shown below (see Figure 17 on page 92) is a minimum inter-physical volume allocation with all the physical partitions for the logical volume on one physical volume:

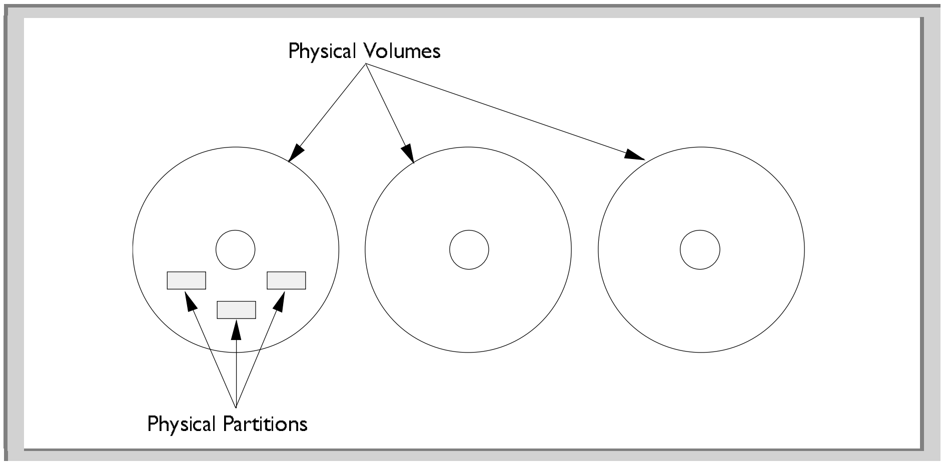


Figure 17. Minimum inter-physical volume allocation policy

The maximum setting, considering other constraints, spreads the physical partitions of the logical volume as evenly as possible over as many physical volumes as possible. This is a performance-oriented option because spreading the physical partitions over several physical volumes tends to decrease the average access time for the logical volume. To improve availability, the maximum setting should only be used with mirrored logical volumes.

Shown below (see Figure 18 on page 93) is a maximum inter-physical volume allocation with all the physical partitions for the logical volume spread over many physical volumes:

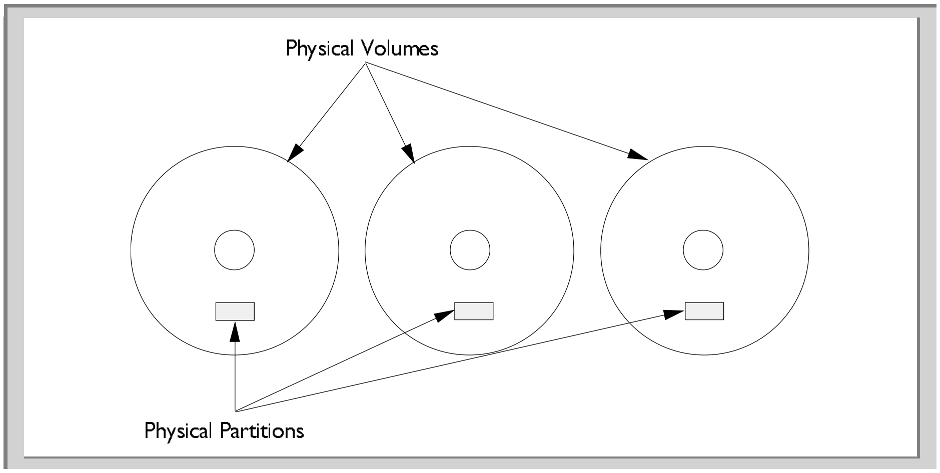


Figure 18. Maximum inter-physical volume allocation policy

These definitions are also applicable when extending or copying an existing logical volume. The allocation of new physical partitions is determined by your current allocation policy and where the existing used physical partitions are located.

Inter-physical volume settings for logical volume copies

The allocation of a single copy of a logical volume on physical volume is fairly straightforward. When you create mirrored copies, however, the resulting allocation is somewhat complex. The figures that follow show minimum and maximum inter-physical volume (`Range`) settings for the first instance of a logical volume along with the available `Strict` settings for the mirrored logical volume copies.

For example, if there are mirrored copies of the logical volume, the minimum setting causes the physical partitions containing the first instance of the logical volume to be allocated on a single physical volume, if possible. Then, depending on the setting of the `Strict` option, the additional copy or copies are allocated on the same or on separate physical volumes. In other words, the algorithm uses the minimum number of physical volumes possible within the constraints imposed by other parameters, such as the `Strict` option, to hold all the physical partitions.

Setting the `Strict` parameter to `y` means that each copy of the logical partition will be placed on a different physical volume. Setting the `Strict` parameter to `n` means that the copies are not restricted to different physical volumes.

Setting the `Strict` parameter to `s` means that the partitions allocated for one mirror cannot share a physical volume with the partitions from another mirror

Note

If there are fewer physical volumes in the volume group than the number of copies per logical partition you have chosen, you should set `Strict` to `n`. If `Strict` is set to `y` or `s`, an error message is returned when you try to create the logical volume.

Shown in Figure 19 is a minimum inter-physical volume allocation for a mirrored logical volume with strictness set to "n". All physical partitions for the logical volume are on one physical volume.

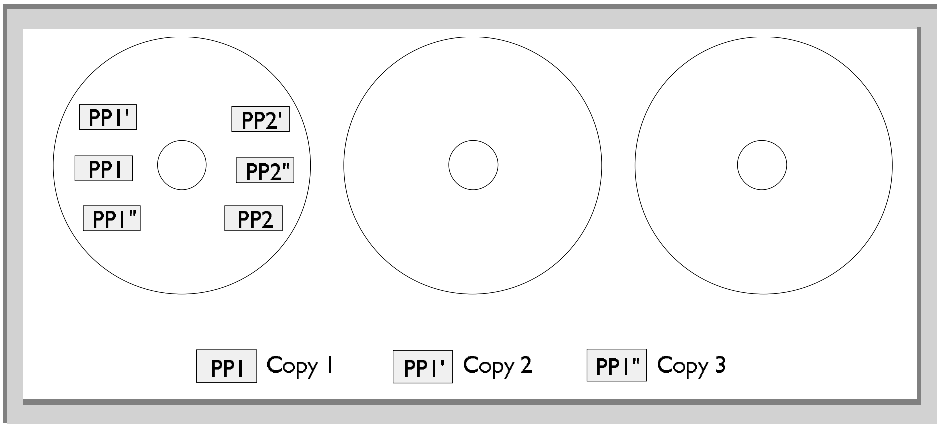


Figure 19. Minimum inter-physical volume policy with strictness set to 'no'

Shown below (see Figure 20 on page 95) is a minimum inter-physical volume allocation for a mirrored logical volume with strictness set to "y". Each copy of a logical partition is on a different physical volume, though not all copies are on the same one.

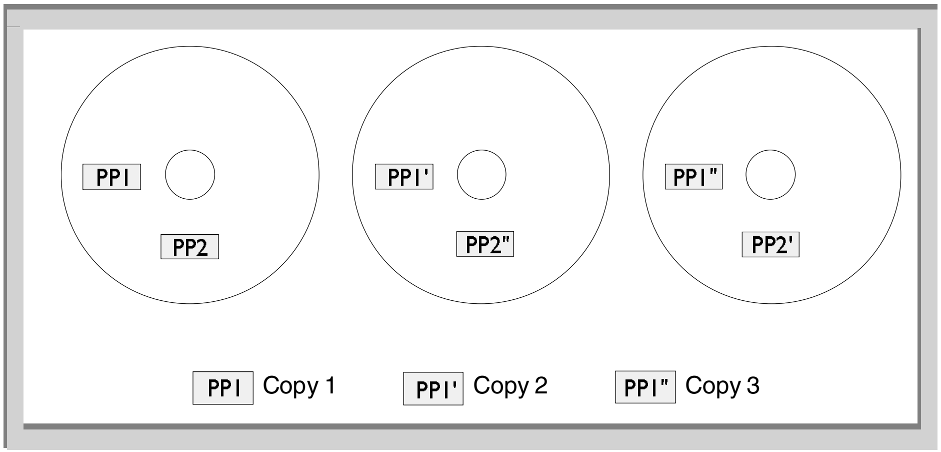


Figure 20. Minimum inter-physical volume policy with strictness set to 'yes'

Shown in Figure 21 is a minimum inter-physical volume allocation for a mirrored logical volume with strictness set to "s". Each copy of a logical partition is on its own physical volume.

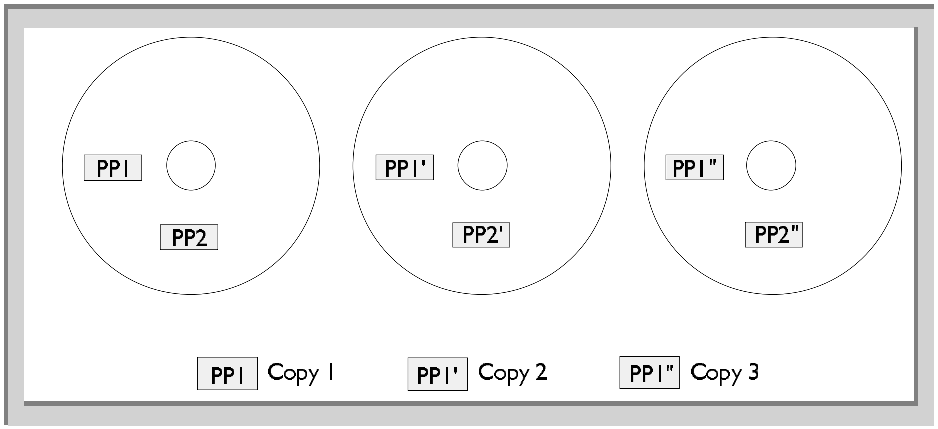


Figure 21. Minimum inter-physical volume policy with strictness set to 'strict'

Shown in Figure 22 on page 96 is a maximum inter-physical volume allocation for a mirrored logical volume with strictness set to "n". Each physical partition is on a separate physical volume with its copies.

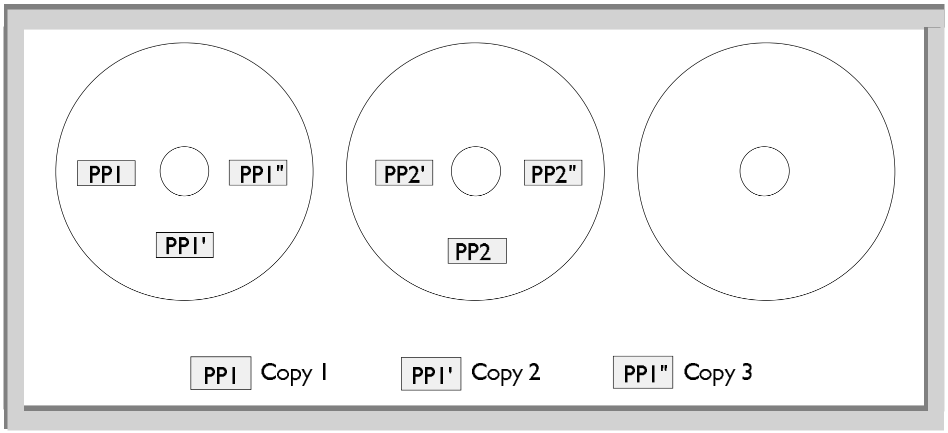


Figure 22. Maximum inter-physical volume policy with strictness set to 'no'

Shown in Figure 23 is a maximum inter-physical volume allocation for a mirrored logical volume with strictness set to "y". Each physical partition is on a separate physical volume, separate to its copies.

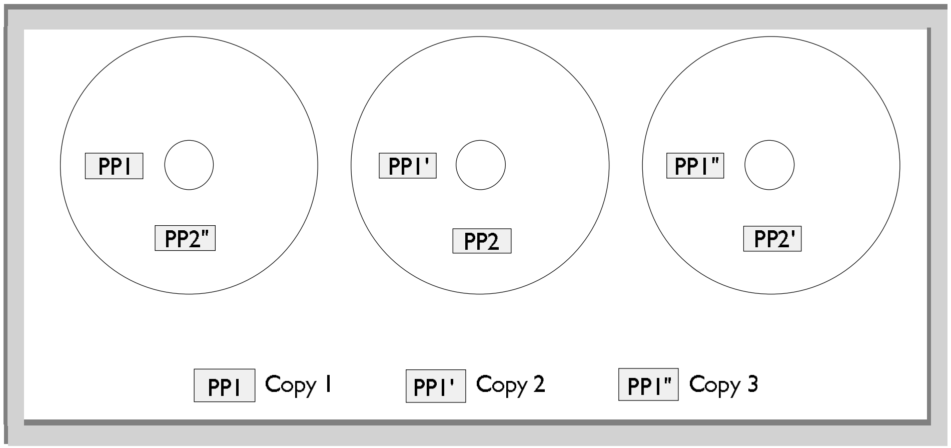


Figure 23. Maximum inter-physical volume policy with strictness set to 'yes'

Shown Figure 24 on page 97 is a maximum inter-physical volume allocation for a mirrored logical volume, with strictness set to "s". Each physical partition is on a separate physical volume; no copy is on a physical volume with a partition from another copy. That is, primary, secondary, and tertiary mirror copies for one logical volume are not sharing a physical volume.

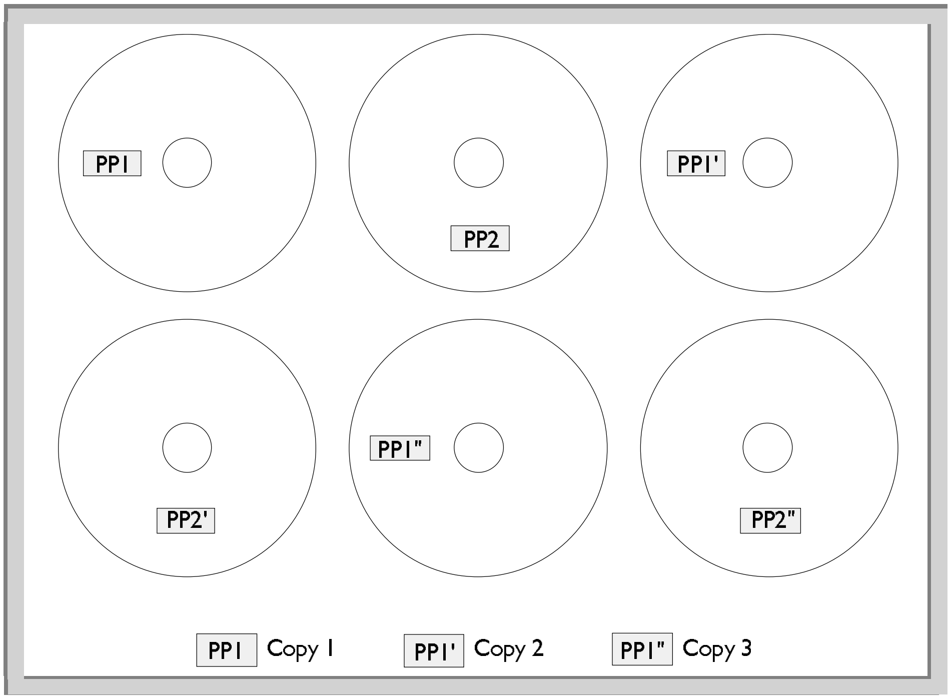


Figure 24. Maximum inter-physical volume policy with strictness set to 'strict'

1.4.2.4 Choose an intra-physical volume allocation policy

If the physical volume is less than 4 GB in capacity, then the closer a given physical partition is to the center of a physical volume, the lower the average seek time because the center has the shortest average seek distance from any other part of the physical volume. If the physical volume is greater than 4 GB, then, due to packing and angular velocity, the outer edge is the better region for performance. There is also the issue of the location of the MWC cache (see Chapter 7, "Performance" on page 303).

The file system log is a good candidate for allocation at the center/outer edge of a physical volume because it is used by the operating system so often. At the other extreme, the boot logical volume is used infrequently and, therefore, should be allocated at the edge or middle of the physical volume.

The general rule, then, is that the more I/Os, either absolutely or during the running of an important application, the closer to the center of the physical volumes the physical partitions of the logical volume should be allocated if the physical volume is less than 4 GB. If greater than 4 GB, then the outer edge should be used. This rule has two important exceptions:

1. Logical volumes that contain large, sequential files should be at the edge because sequential performance is better there (there are more blocks per track at the edge than farther in).
2. Mirrored logical volumes with Mirror Write Consistency (MWC) set to ON should be at the outer edge because that is where the system writes MWC data. If mirroring is not in effect, MWC does not apply and does not affect performance. Otherwise, see Chapter 7, “Performance” on page 303, and "Performance Implications of Disk Mirroring" in the section on performance-related installation guidelines in the *AIX Performance Tuning Guide*, SC23-2365.

The intra-physical volume allocation policy choices are based on the five regions of a physical volume where physical partitions can be located. These regions are discussed in Chapter 1.2.1.1, “The structure of a physical volume” on page 7.

1.4.2.5 Combining Allocation Policies

If you select inter-physical volume and intra-physical volume policies that are not compatible, you may get unpredictable results. The system will assign physical partitions by allowing one policy to take precedence over the other. For example, if you choose an intra-physical volume policy of center and an inter-physical volume policy of minimum, the inter-physical volume policy will take precedence. The system will place all of the partitions for the logical volume on one physical volume if possible, even if the partitions will not all fit into the center region. Make sure you understand the interaction of the policies you choose before implementing them.

1.4.2.6 Using map files for precise allocation

If the default options provided by the inter- and intra-physical volume policies are not sufficient for your needs, consider creating map files to specify the exact order and location of the physical partitions for a logical volume.

Chapter 1.2.8.5, “The logical volume map file” on page 51 gives more information on the use of the map file.

1.4.2.7 Developing a striped logical volume strategy

Striped logical volumes are used for large sequential file systems that are frequently accessed and performance-sensitive. Striping is intended to improve performance. On versions of AIX below 4.3.3, the availability of the striped logical volume is low compared with a mirrored logical volume, as both striping and mirroring are not supported. However with AIX Version 4.3.3, both mirroring and striping are supported.

1.4.2.8 Determine a write-verify policy

Using the write-verify option causes all write operations to be verified by an immediate follow-up read operation to check the success of the write. If the write operation is not successful, you will get an error message. This policy enhances availability but degrades performance because of the extra time needed for the read. You can specify the use of a write-verify policy on a logical volume either when you create it (`mklv`) or later by changing it (`chlv`).

1.4.2.9 Implement volume group policies

1. Use the `lspv` command to check your allocated and free physical volumes. In a standard configuration, the more physical volumes that make up a quorum volume group the better the chance of the quorum remaining when a disk failure occurs. In a non-quorum group, a minimum of two physical volumes must make up the volume group.
2. To ensure a quorum,
   - Add one or more quorum buster physical volumes
   - switch off quorum for the volume group
3. The common configuration is a single volume group that includes multiple physical volumes attached to the same physical volume adapter and other supporting hardware. Reconfiguring the hardware is an elaborate step. Separate hardware is only necessary if your site requires high availability.

1.5 Limits

Table 12 on page 99 lists the limitations for the Logical Volume Manager components across the various versions of AIX.

Table 12. Logical volume limitations

Limitations for Logical Storage Management				
	< 4.3.1	4.3.1	4.3.2 / 3	big vgda
Volume groups / system	255	255	255	255
Physical volume / volume group	32	32	128	128
Physical partition / physical volume	1 016	32 512[a]	32 512	130 048
Logical volume / volume group	256	256	256	512[b]
Logical partition / Logical volume	32 512	32 512	130 048	130 048
Physical partition size = 2^n, where n	$20 \le n \le 28$	$20 \le n \le 30$	$20 \le n \le 30$	$20 \le n \le 30$
Logical to physical partition mapping	1-3	1-3	1-3	1-3

Limitations for Logical Storage Management				
	< 4.3.1	4.3.1	4.3.2 / 3	big vgda
Physical block (sector) size	512	512	512	512
Logical volume size	2GB	1TB	1TB	1TB
Mirrored	Yes	Yes	Yes	Yes
Striped	No	Yes	Yes	Yes
Striped and mirrored	No	No	Yes	Yes
File system size	2GB	1024GB	1024GB	1024GB

a. With AIX 4.3.1 one was able to increase the maximum number of physical partitions by factors of 1016. This however implies that the number of disks that can be added to the volume group is reduced, as the maximum number of Logical Partitions per Logical Volume is not changed.

b. With -G flag

Chapter 2. Mirroring

The basic concept of mirroring is simple – keep the data in different locations to eliminate the possibility of data loss upon disk block failure situation. From the high-availability point of view, each mirrored data should be located on separate physical disks, and using separate I/O adapters. Furthermore, put each disk into a separate disk drawer to protect the data loss upon power failure situation (but the degree of high-availability depends on the physical peripheral availability and costs).

This chapter describes the mirroring function (also known as RAID 1 technology) of the AIX Logical Volume Manager, its principle, and its concepts. It will also provide a practical example on how to create a mirrored logical volume. The particular case of mirroring the root volume group is also included. The performance issues of mirrored logical volumes are covered in detail in Chapter 7, “Performance” on page 303.

2.1 Principles

If you using the AIX LVM mirroring function, here are some basic rules that you must be aware of:

- Only logical volumes are mirrored.
- Each logical volume may have two or three mirrored instances (called *copy*).
- Keep it simple.

The following sections provide explanations about these principles.

2.1.1 Only logical volumes are mirrored

In AIX, the mirroring function is provided by its LVM components. Reading the section “Mirroring of the rootvg” on page 132 may let you think that entire disk are mirrored. This is not true. The AIX mirror function does not apply to a physical disk, only to logical volumes. This is the most important principle to understand for the AIX LVM mirroring function.

2.1.2 Each logical volume can have up to three copies

The AIX Logical Volume manager allows a total of three copies of the data; the original one plus two copies.

So, each logical partitions (LP) belonging to the mirrored logical volume may have up to three copies. These instances are called physical partitions (PP). To correlate one logical partition to the physical partitions which resides on the physical volumes, we use a mechanism called *mapping*. Each logical partition and physical partition have an instance number for unique identification.

For example, the logical volume hd9var is created with a four logical partitions size (See Figure 25). In this example, this logical volume is allocated four physical partitions on both physical volumes (shown as a hatched rectangle portion in the figure). The logical partition 1 is mapped to the physical partition 220 on the physical volume 1, and to the physical partition 190 on physical volume 2.

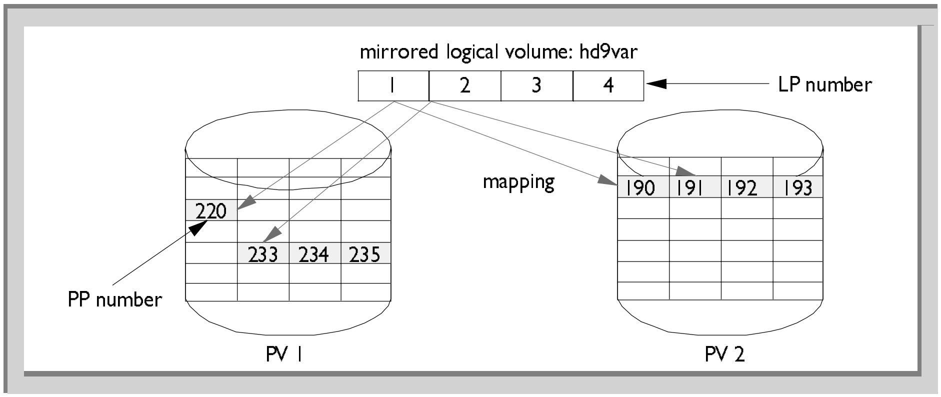


Figure 25. Mapping - Relationship between logical partition and physical partitions

To confirm this mapping between the logical partition and the physical partitions, you can use the `lslv` command with the `-m` option as shown below:

```
root@pukupuku:/ [110] # lslv -m hd9var
hd9var:/var
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0220 hdisk0            0190 hdisk1
0002  0233 hdisk0            0191 hdisk1
0003  0234 hdisk0            0192 hdisk1
0004  0235 hdisk0            0193 hdisk1
```

In this example, the logical volume hd9var has two copies. The logical partition 0001 of this logical volume is mapped to the physical partition 0220 on hdisk0 and to the physical partition 0190 on hdisk1.

2.1.3 Keep it simple

If you create a lot of logical volumes in a volume group, you will make things too complex and your configuration will rapidly become unmanageable. That is why you should keep it simple and easy to understand. The following examples provide some rules that enables you to *keep it simple*.

- Minimize the number of logical volumes in a volume group as much as possible, there are some situations where you still have to do that (see Figure 26).

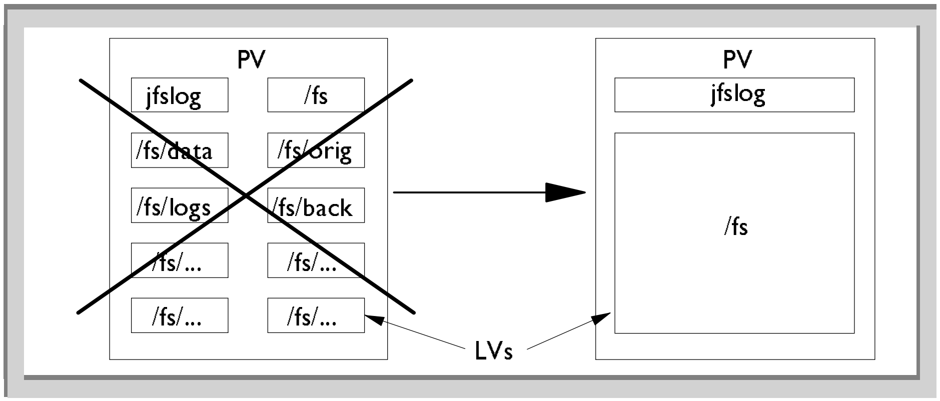


Figure 26. Minimize the number of logical volumes in a volume group

- Keep your volume group as small as possible, in other words, avoid creating a logical volume spread over more than two physical volumes (shown as LV2 in Figure 27). One major exception would be if your logical volume size does not fit into one physical volume. If you create huge volume groups involving a lot of disks, you will loose flexibility. For example, dedicating a volume group to the operating system, then other volume groups to application and user data, allows you to migrate the system without having to restore users' data or to vary offline some applications and not the users' data.

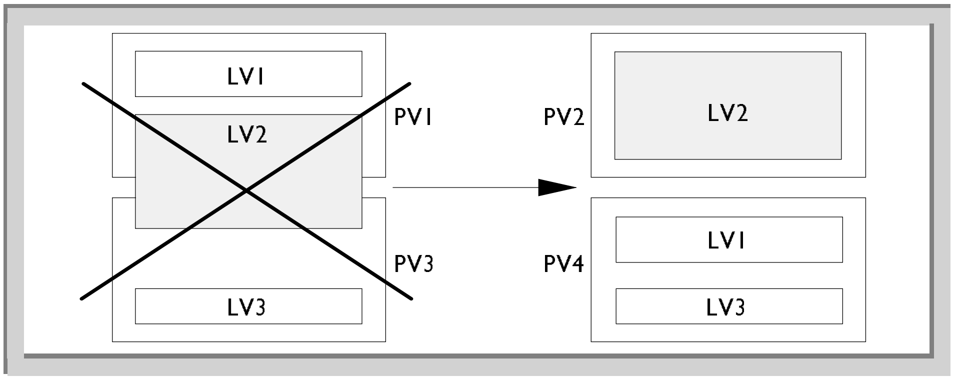


Figure 27. Avoid logical volumes spread over multiple physical volumes

- Avoid creating scattered physical partitions in logical volumes. An example is shown with various patterned rectangles scattered across the physical volumes in the left rectangle in Figure 28. Usually, scattered logical volumes are produced by the JFS extension activity. This dynamic extension of journaled file systems is one of the outstanding features of AIX but it may sometimes impact manageability and performance. From that point of view, the best discipline would be to create the logical volume with the appropriate size at installation time and not extend it further (shown in the various patterned rectangles on the right side in Figure 28). The `reorgvg` command, discussed in the Chapter 5, “The AIX journaled file system” on page 233, is provided to solve this problem.

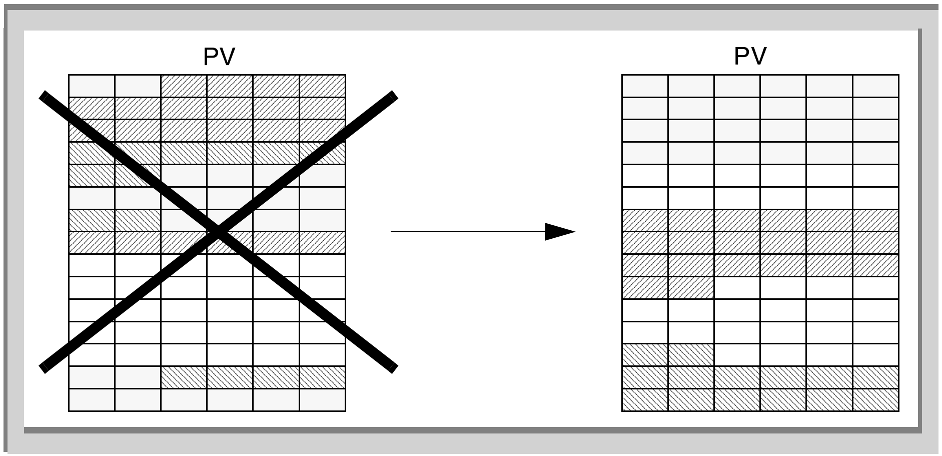


Figure 28. Do not scatter physical partitions of logical volumes

2.2 Concepts

This section provides detailed information about some important points. It presents the allocation policy, scheduling policy, and integrity concepts.

2.2.1 The allocation policy

Each logical volume can have up to three copies. To create a mirror for a logical volume, you must have physical partitions allocated separately on several physical disks that are part of a volume group. There are some considerations you must be aware of in order to understand the allocation scheme for physical partitions of mirrored logical volumes.

2.2.1.1 Strict allocation policy

The default behavior for AIX creating a mirror for a logical volume is to try and place all the physical partitions containing the same data on separate physical volumes. In our example shown in Figure 29, all the physical partitions of the mirrored logical volume are placed on separate physical volumes (PV1 and PV2). This is a valid and acceptable allocation for the AIX LVM with the default strict allocation policy.

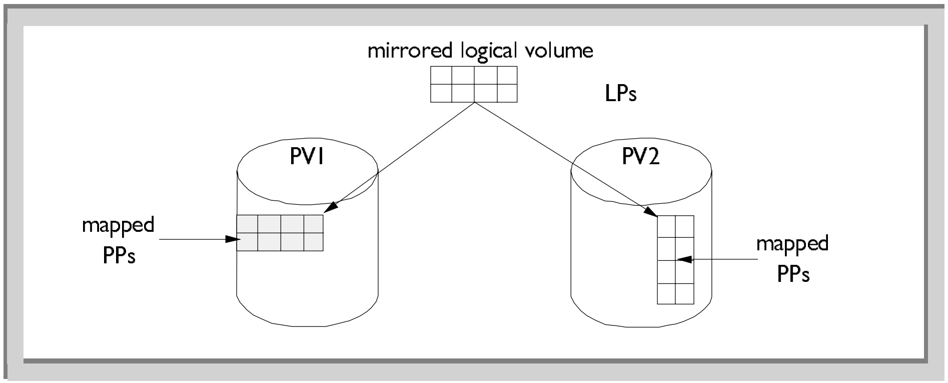


Figure 29. Default allocation of physical partitions for mirrored logical volume

Suppose that before you decide to create the mirror for the logical volume, there is already not enough room on the second physical disk to allocate the necessary copy of the logical partitions (see Figure 30 on page 106). The attempt to create the mirror will fail with the following error message.

```
root@pukupuku:/ [103] # mklvcopy hd6 2 hdisk0
0516-404 allocp: This system cannot fulfill the allocation request.
        There are not enough free partitions or not enough physical volumes
        to keep strictness and satisfy allocation requests. The command
        should be retried with different allocation characteristics.
```

The reason for this failure is the strict allocation policy of the AIX LVM. With a mirroring where several copies of the same data are used, a failure of that disk would result in the complete loss of the mirrored logical volume. In most circumstances, this is not a safe configuration, therefore, by default, the AIX LVM does not allow you to place more than one copy of a logical partition on a given physical volume.

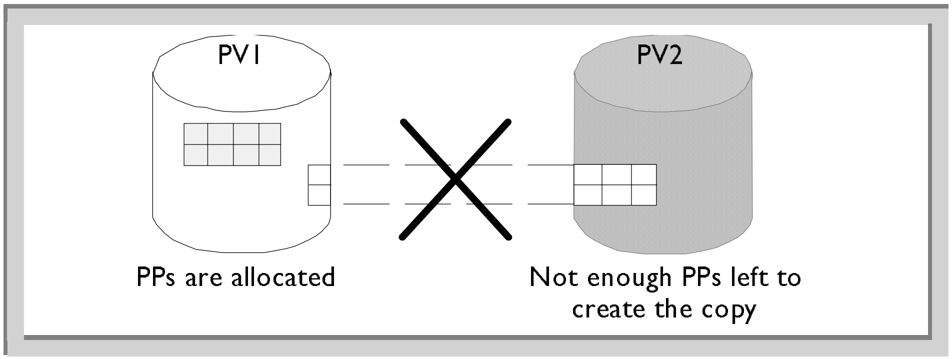


Figure 30. Undesirable allocation of physical partitions for mirrored logical volume

Though this allocation policy exists, if you still want to create the mirror of a logical volume on the same physical volume (at your own risk) you can override this policy by using the `-s` option of the `mklv` command:

```
# mklv -s n -y notstrictlv 10 mirrorvg
```

The following descriptions are from the `mklv` command reference in AIX Version 4.3.3.

`-s Strict` Determines the strict allocation policy. Copies of a logical partition can be allocated to share or not to share the same physical volume. The `Strict` variable can be one of the following:

`y` Sets a strict allocation policy, so copies for a logical partition cannot share the same physical volume. This is the default for the `-s` flag.

`n` Does not set a strict allocation policy, so copies for a logical partition can share the same physical volume.

s Sets a super strict allocation policy, so that the partitions allocated for one mirror cannot share a physical volume with the partitions from another mirror.

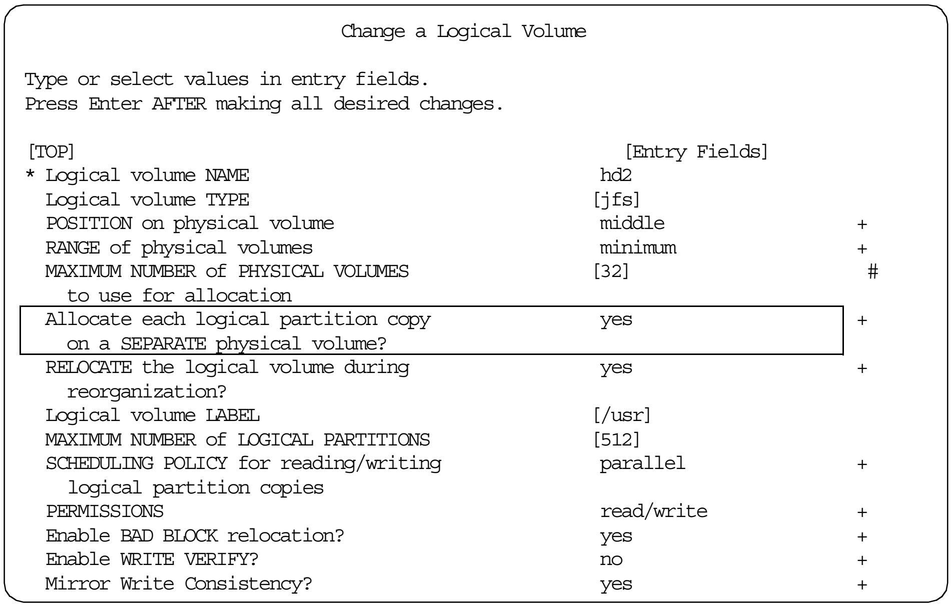
Note

The super strict allocation policy is newly introduced in AIX Version 4.3.3 to support the mirror and stripe function. For details about this allocation type, refer to 3.3, “The mirror and stripe function” on page 162.

Actually, the default behavior of the allocation policy is determined by an attribute of the logical volume you are trying to make a mirror. To check this attribute, you can use the following SMIT dialog:

```
# smit -C lv
    Set Characteristic of a Logical Volume
           Change a Logical Volume
```

Choose the appropriate logical volume name as LOGICAL VOLUME name.



```
                              Change a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
* Logical volume NAME                                 hd2
  Logical volume TYPE                                [jfs]
  POSITION on physical volume                         middle          +
  RANGE of physical volumes                           minimum         +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                 [32]              #
    to use for allocation
  Allocate each logical partition copy                yes             +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                  yes             +
    reorganization?
  Logical volume LABEL                               [/usr]
  MAXIMUM NUMBER of LOGICAL PARTITIONS               [512]
  SCHEDULING POLICY for reading/writing               parallel        +
    logical partition copies
  PERMISSIONS                                         read/write      +
  Enable BAD BLOCK relocation?                        yes             +
  Enable WRITE VERIFY?                                no              +
  Mirror Write Consistency?                           yes             +
```

Otherwise, you can examine this from the command line. The field showing the allocation policy is shown in the rectangle in the following example.

```
root@pukupuku:/ [132] # lslv hd2
LOGICAL VOLUME:     hd2                    VOLUME GROUP:   rootvg
LV IDENTIFIER:      00702137e70a7982.5     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                152                    PPs:            152
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        /usr                   LABEL:          /usr
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
```

2.2.1.2 Mirroring with exact mapping

Within a new or empty volume group, creating a mirror usually results in a mirroring with *exact mapping* (this comes from the algorithm used but there is no guarantee that this happens. Using the map files is the only safe way). Exact mapping is the function that allocates logically and/or physically identical positions for physical partitions on each physical volume used to create the mirror, and is divided into two cases; logically identical allocation and physically identical allocation. The following section describes these concepts.

Logically identical allocation

If you use different size disks for mirroring, you may have a mirrored logical volume with a logically identical location using the exact mapping function. For example, let us take a volume group composed of two different size physical disks and that have a 16 MB size physical partition. The numbering scheme of the physical partition may be the same, but the physical location of the partition on the disk will vary. The physical partition number 233 can be located in the center of disk A and middle edge of disk B, ruining your effort to reach best performance.

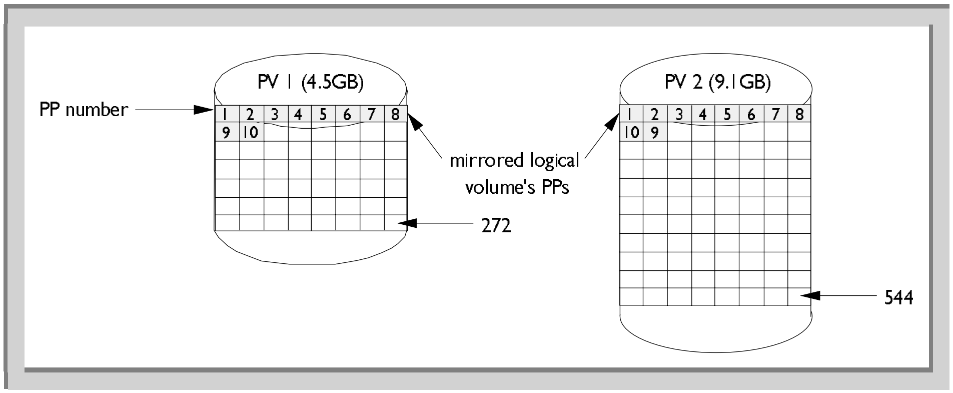


Figure 31. Exact mapping - Logically identical allocation

Physically identical allocation

If you use the same disks type (having the same physical size) for mirroring, you will obtain a mirrored logical volume with physically identical allocation using the map files. For example, the volume group is composed of two same size physical volumes and has 8 MB physical partition size (see Figure 32). This is the best case (and probably most efficient from the performance and system management perspective) for the mirrored logical volumes with exact mapping.

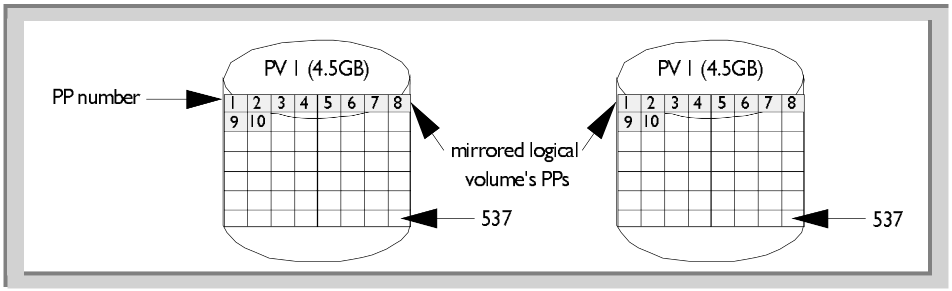


Figure 32. Exact mapping - Physically identical allocation

Mirroring without exact mapping function

If you don’t use an empty volume group and if you already have a logical volume occupying the corresponding physical locations on the second disk, you remain with the alternative of not having an exact mapping situation and locating the partitions in a different region of the disk.

Note that the mirror configuration of this logical volume is still valid from the data availability perspective.

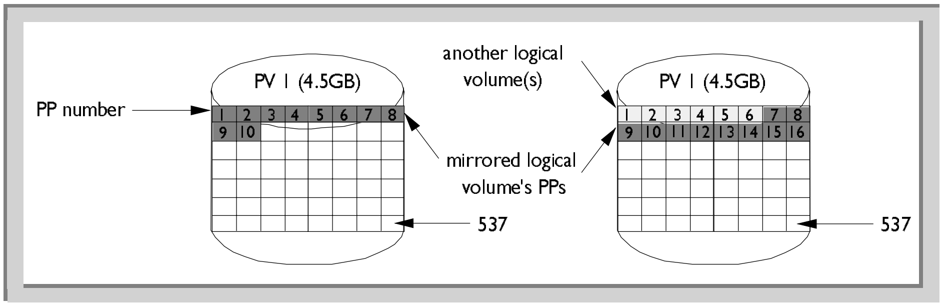


Figure 33. Mirroring without exact mapping

There is a particular case where you cannot reach an exact mapping situation. This is when you are using different size disks. Imagine that you have a volume group made of one 4.5 GB disk and one 9.1 GB disk. You create your first copy of the logical volume on the 9.1 GB disk using the physical partitions 500 to 510. When to try to create a copy of this logical volume on the 4.5 GB disk, you realize that there is no physical partition 500 on that disk. Its size limits the number of partitions to 272. This would reinforce the rule of using comparable disks to create mirrored logical volumes.

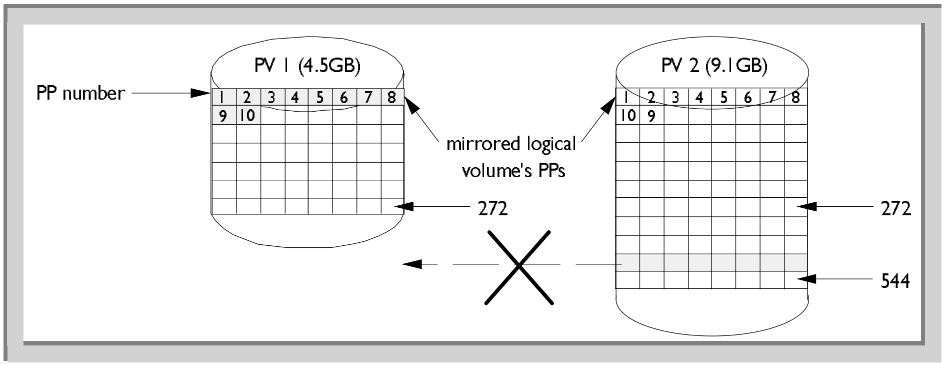


Figure 34. Allocation impossible with exact mapping

2.2.1.3 Map files

To allow the users to fully control the allocation algorithm of physical partitions for specified logical volumes, the AIX LVM provides the *map file* facility with some AIX LVM high-level commands. The map file is a text file that specifies how the AIX LVM should allocate the physical partitions of the specified physical volume. The commands that provide this map file mechanism are `mklv`, `mklvcopy`, and `extendlv`. Another command that uses this facility (but does not take a map file name as a command line option) is `mirrorvg`. The `savevg` and `mksysb` commands also deal with the map files.

Note

`mirrorvg` is the only command providing a real exact mapping function. The `-m` flag allows you to create the mirror of a volume group that has for every logical volume the same physical partitions.

2.2.2 The scheduling policy

If you have multiple copies of a specific logical volume on separate physical disks, how do you decide how an I/O request (for example, read or write system call) is made? On which disk do you write first? When is the request really finished, after the first disk has been written or all of them have? The answer is included in the concept called *scheduling policy*, and is determined by an attribute of the logical volume. There are two types of scheduling policy, *parallel* (the AIX default) and *sequential*.

In general discussion about the mirroring mechanism, we won't distinguish between the original copy and its mirrors; that is, the first one and the ones created after from that master. The exception to this is when we discuss the read, write request order in sequential scheduling policy. In sequential scheduling policy, there are distinct *primary* copy and *secondary* copies.

The default behavior of this scheduling policy is determined by an attribute of the logical volume considered. To check the value of this attribute, you can use the following SMIT dialog:

```
# smit -C lv
   Set Characteristic of a Logical Volume
       Change a Logical Volume
```

Choose the appropriate logical volume name as LOGICAL VOLUME name.

```
                              Change a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                [Entry Fields]
* Logical volume NAME                                hd2
  Logical volume TYPE                               [jfs]
  POSITION on physical volume                        middle            +
  RANGE of physical volumes                          minimum           +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                [32]                #
    to use for allocation
  Allocate each logical partition copy               yes               +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                 yes               +
    reorganization?
  Logical volume LABEL                              [/usr]
  MAXIMUM NUMBER of LOGICAL PARTITIONS              [512]
  SCHEDULING POLICY for reading/writing              parallel          +
      logical partition copies
  PERMISSIONS                                        read/write        +
  Enable BAD BLOCK relocation?                       yes               +
  Enable WRITE VERIFY?                               no                +
  Mirror Write Consistency?                          yes               +
```

The scheduling policy is shown in the rectangle in the previous figure.

Otherwise, you can examine this from the command line (the attribute is also shown in the following rectangle):

```
root@pukupuku:/ [132] # lslv hd2
LOGICAL VOLUME:     hd2                    VOLUME GROUP:   rootvg
LV IDENTIFIER:      00702137e70a7982.5     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY: parallel
LPs:                152                    PPs:            152
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        /usr                   LABEL:          /usr
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
```

2.2.2.1 The parallel scheduling policy

The majority of mirrors created on AIX have the parallel scheduling policy (it is the default value for a newly created logical volume). With the parallel scheduling policy, there is no primary or secondary mirror. All copies in a mirror set are just referred to as copy, regardless of which one was created first. Since the user can remove any copy from any disk, at any time, there can be no ordering of copies.

With the parallel scheduling policy, all copies are considered equal. Thus, when a read request arrives at the LVM, there is no first or favorite copy that is accessed for the read. A search is done on the request queues for the drives which contain the mirror physical partition that is required. The drive that has the fewest pending requests is picked as the disk drive that will service the read request.

On write requests, the LVM driver broadcasts to all drives that have a copy of the physical partition that needs updating. Only when all write requests return is the write considered complete and the write complete message is returned to the calling program (see Figure 35).

Mirroring can improve the read performance of a system, but the price to pay is the write performance. Of the two mirroring scheduling policies, parallel and sequential, parallel is the best in terms of disk I/O.

In parallel scheduling policy, when a read request is received, the LVM device driver looks at the queued requests (read and write) and finds the disk with the smallest number of requests waiting to execute.

If mirroring can provide an improvement for the read requests, the price to pay is a longer cycle to write the data. When the LVM disk driver receives a write request (shown as step 1 in Figure 35 on page 114), it must now perform separate writes (shown as step 2 in Figure 35), then each disk must complete its own write operation (shown as step 3 in Figure 35) before the LVM device driver considers a write request as complete and returns write acknowledgement to the caller (shown as step 4 in Figure 35).

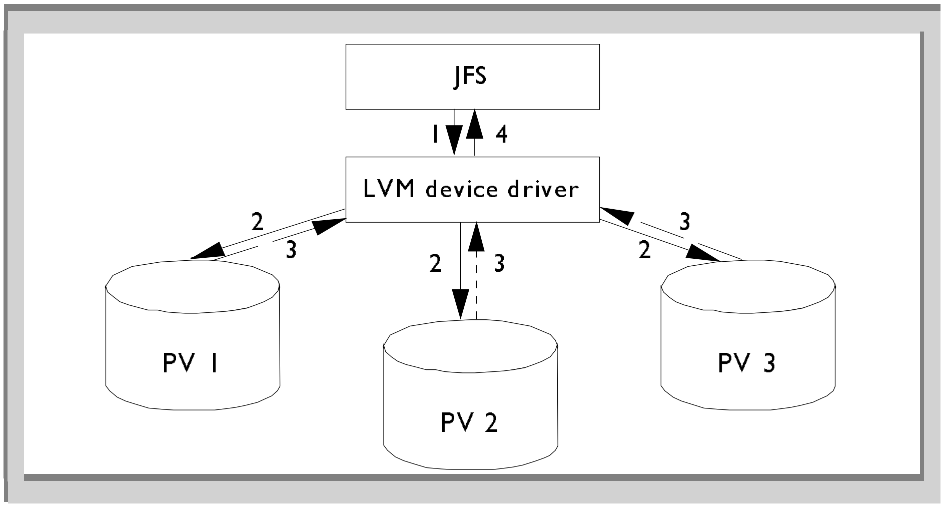


Figure 35. Parallel scheduling policy - Write sequence

2.2.2.2 The sequential scheduling policy

The sequential scheduling policy (as opposed to parallel) is based on the idea of an order within the mirrors. All read and write requests (shown as 1 in Figure 36 on page 115) first go through a primary copy that services the request in sequential scheduling policy (shown as 2 and 3 in Figure 36). If the request is a write, then the write request is propagated sequentially to the secondary drives (shown as 4 and 5 in Figure 36). Once the secondary (and tertiary, if applicable) drives have serviced the same write request, then the LVM device driver will consider the write request complete (see Figure 36 on page 115).

If the request is a read, then the read request is forwarded to the primary drive. If the read request is successfully returned from the primary drive, it satisfies the read request from the caller and returns. Otherwise, the secondary (or tertiary, if applicable) drive is examined.

Obviously, there is a performance disadvantage to using the sequential scheduling policy rather than the parallel scheduling policy in mirroring. The question arises why anyone would be interested in using sequential scheduling policy. The answer is that since there is a definite and defined *primary copy*, this will be the first disk always read during a reboot. Thus, it will be the *source* mirror copy used when mirror re-synchronization occurs. The implication of this will become clear in “Mirror Write Consistency” on page 117.

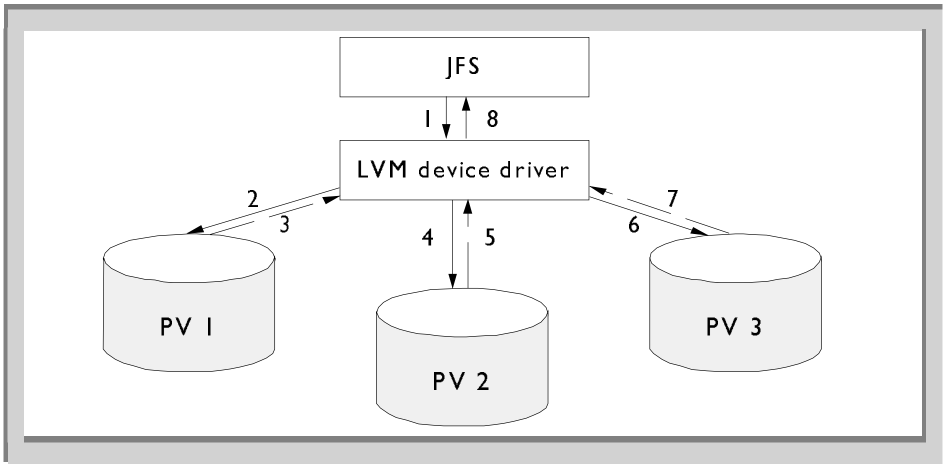


Figure 36. Sequential scheduling policy - Write sequence

2.2.3 Availability

The AIX LVM allows up to three copies for a logical volume and the sequential or parallel scheduling policy. Mirrors improve data availability on a system by providing more than one source of identical data. With multiple copies of a logical volume, if one copy can not provide the data, one or two secondary copies may be accessed to provide the desired data.

But how does the AIX LVM maintain the integrity of the data that resides in mirrored logical volumes after a system crash, for example? If there is no protection mechanism, you probably have a mirrored logical volume that has a data divergence. This section focuses on the mechanism provided by AIX LVM to guarantee the data integrity of a mirrored logical volume.

2.2.3.1 Quorum checking

As discussed in 1.2.7.11, “Quorum” on page 39, quorum checking is the voting that goes on between disks in a volume group to see if a majority of disks exist to form a quorum that will allow the disks in a volume group to become and stay activated. LVM runs many of its commands and strategies based on having the most current copy of data. Thus, it needs a method to compare data on two or more disks and figure out which one contains the most current information.

So, if you have a volume group that has more than two physical volumes, you may encounter a strange situation. For example, a volume group is composed of four physical volumes, and each physical volume has a copy of a logical volume (see Figure 37).

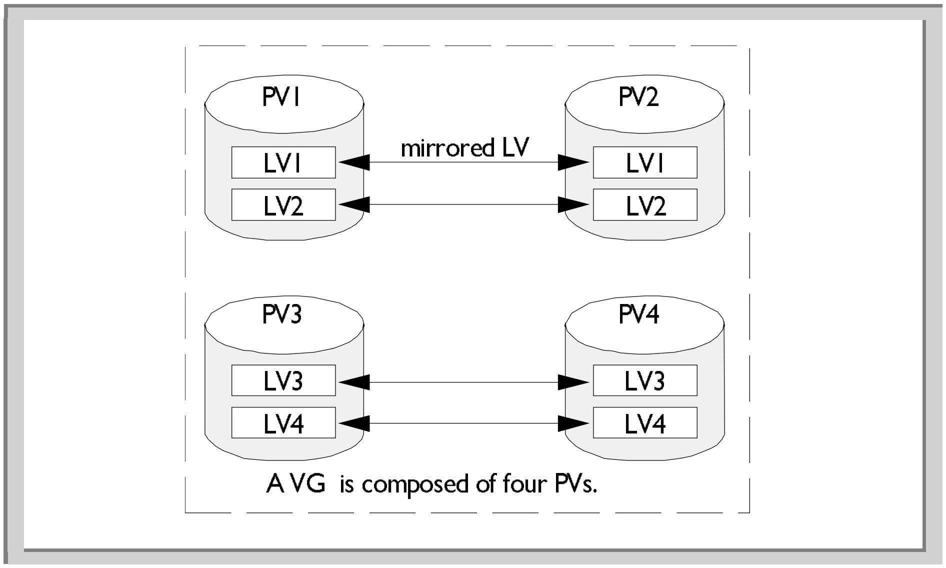


Figure 37. Quorum checking dilemma

Suppose that you have not disabled the quorum for this volume group. In this situation, should PV 1 and PV 3 fail, then this volume group will be closed, even if valid copies of the PV1 and PV3 are still accessible on PV2 and PV4. So, you decide to disable the quorum for this volume group. Now, it is PV 1 and PV 2 that fail, the volume group will not be closed, and you will have a data accessibility problem on the logical volumes LV1 and LV2. This is a quorum dilemma. To avoid this dilemma, you have to remember the last principle: *Keep it simple*. In this case, you could create two volume groups that are composed of two physical volumes each, instead of one big volume group.

2.2.3.2 Staleness

The idea of a mirror is to provide an alternate, physical copy of information. If it is not possible to write to one copy at one point, usually due to a disk failure, then the AIX LVM refers to that copy of the mirror as going *stale*. Staleness is determined by the LVM device driver when a request to the disk device driver returns with a certain type of error. When this occurs, the AIX LVM device driver notifies the *VGSA* of a disk that a particular physical partition on that disk is stale. This information will prevent further read or writes from being issued to physical partitions defined as stale by the VGSA of that disk.

Additionally, when the disk once again becomes available (suppose it had been turned off accidentally), the synchronization code knows exactly which physical partitions must be updated, instead of defaulting to the update of the entire disk. Certain high-level commands will display the physical partitions and their stale condition so that the user can be informed of which disks may be experiencing a physical failure.

Following is a example of how to examine what partitions are stale. The `lsvg -l` command shows that the logical volume hd4, which houses the root file system that has stale partitions. The `lslv -p` command shows that the logical partitions number 1 and 2 that reside on physical partitions number 8 and 9 (the far right column shows the physical partition number range) are marked stale.

```
root@pukupuku:/ [104] # lsvg -l rootvg
rootvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
hd5                 boot       1     2     2    closed/syncd  N/A
hd6                 paging     16    32    2    open/syncd    N/A
hd8                 jfslog     1     2     2    open/syncd    N/A
hd7                 sysdump    4     4     1    open/syncd    N/A
hd4                 jfs        2     4     2    open/stale    /
hd2                 jfs        152   304   1    open/syncd    /usr
hd9var              jfs        4     8     2    open/syncd    /var
hd3                 jfs        4     8     2    open/syncd    /tmp
opt.lotus           jfs        32    64    2    open/syncd    /opt/lotus
netscape            jfs        4     8     2    open/syncd    /usr/netscape
usr.share.man       jfs        40    80    2    open/syncd    /usr/share/man
var.adsmserv        jfs        4     8     2    open/syncd    /var/adsmserv
root@pukupuku:/ [127] # lslv -p hdisk1 hd4
hdisk1:hd4:/
FREE   FREE   FREE   FREE   FREE   USED   USED   0001?  0002?  USED       1-10
USED   USED   USED   FREE   FREE   USED   USED   USED   USED   USED      11-20
(after this line was snipped off)
```

Mirror Write Consistency

Mirror Write Consistency (MWC) identifies which logical partitions may be inconsistent if the system or the volume group is not shut down properly. When the volume group is varied back online for use, this information is used to make logical partitions consistent again.

If a logical volume is using MWC, then requests for this logical volume are held within the scheduling layer until the MWC cache blocks can be updated on the target physical volumes. When the MWC cache blocks have been updated, the request proceeds with the physical data Write operations.

MWC tracks the last 62 writes to mirrored logical volumes within a volume group. If the AIX system crashes, upon reboot the last 62 Logical Track Group (LTG) writes to mirrors are examined and one of the mirrors is used as a *source* to synchronize the mirrors (based on the last 62 disk locations that were written). The LTG is a concept that is used in the write phase. Its size is always 128 KB (32 pages, 1 page being 4 KB). All the write beyond this size is divided into multiple LTG size (see Figure 38).

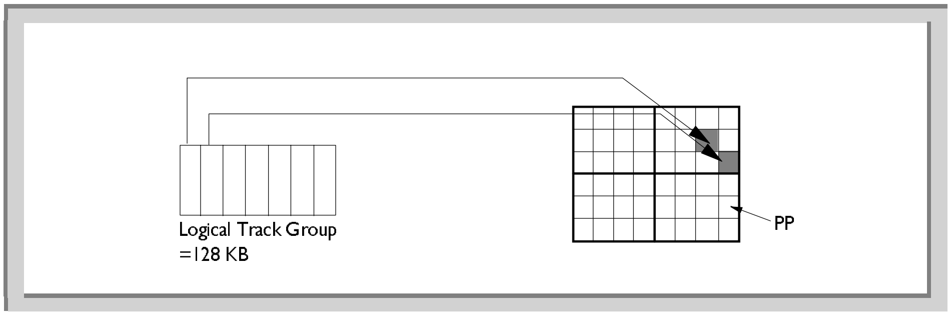


Figure 38. Concept of the logical track group

The MWC is necessary to mirror logical volumes with parallel scheduling policy. In sequentially mirrored scheduling policy logical volumes, the source is always picked to be the *primary* disk. If that disk fails to respond, the next disk in the sequential ordering will be picked as the source copy.

The AIX LVM does not guarantee that the absolute, latest write request completed before a crash will be there after the system reboots. But, it guarantees that the mirrors in parallel scheduling policy will be consistent with each other. From there, the user will be able to realize which writes were considered successful before the system crashed and which writes must be retried.

The point here is not data *accuracy*, but data *consistency*. The use of the primary mirror copy as the source disk is the basic reason that sequential scheduling policy mirroring is offered. Not only is data consistency guaranteed with MWC, but the use of the primary mirror as the source disk increases the chance that all the copies have the latest write that occurred before the mirrored system crashed. Some users turn off MWC for increased write performance or if the software product they use requires that MWC be turned off. You can easily examine this attribute for each logical volume (the MWC attribute is shown in the rectangle):

```
# smit -C lv
    Set Characteristic of a Logical Volume
        Change a Logical Volume
```

Choose the appropriate logical volume name as LOGICAL VOLUME name.

```
                                Change a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
* Logical volume NAME                                 hd2
  Logical volume TYPE                                [jfs]
  POSITION on physical volume                         middle              +
  RANGE of physical volumes                           minimum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                 [32]                  #
    to use for allocation
  Allocate each logical partition copy                yes                 +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                  yes                 +
    reorganization?
  Logical volume LABEL                               [/usr]
  MAXIMUM NUMBER of LOGICAL PARTITIONS               [512]
  SCHEDULING POLICY for reading/writing               parallel            +
    logical partition copies
  PERMISSIONS                                         read/write          +
  Enable BAD BLOCK relocation?                        yes                 +
  Enable WRITE VERIFY?                                no                  +
  Mirror Write Consistency?                          yes                 +
```

Otherwise, you can examine this from the command line (the MWC attribute is also shown in the rectangle):

```
root@pukupuku:/ [132] # lslv hd2
LOGICAL VOLUME:     hd2                    VOLUME GROUP:   rootvg
LV IDENTIFIER:      00702137e70a7982.5     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                152                    PPs:            152
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        /usr                   LABEL:          /usr
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
```

In the case where MWC can not guarantee mirror consistency, the user should run a forced volume group sync (`syncvg -f -v <vgname>`) immediately after a system crash.

The following section explains how the MWC works to guarantee data consistency.

- Before a write to the mirrored logical volume

Suppose that we have the following situation: Only even numbered Logical Track Groups have been the last 62 writes to the mirrored logical volume (see Figure 39). All disks in the volume group, that store a copy of a logical volume for which MWC is enabled track this MWC table.

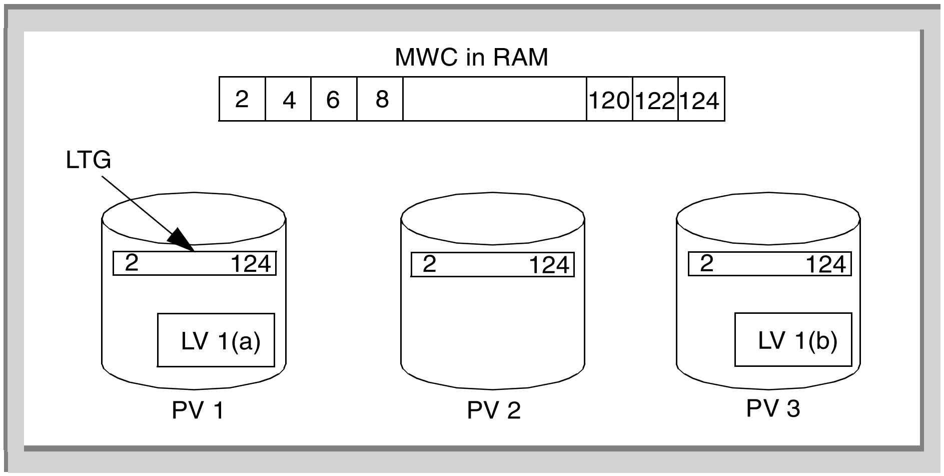


Figure 39. Before the write to the mirrored logical volume

- After a write to the mirrored logical volume

  A write to an odd LTG (9) causes the oldest entry in the MWC (2) to be deleted and the entry to be entered in RAM (see Figure 40 on page 121). Then, the MWC table must be written to the relevant disks in the volume group.

Note

This replacement and update only occurs if the new LTG number is not already one of the last 62 entries in the MWC table.

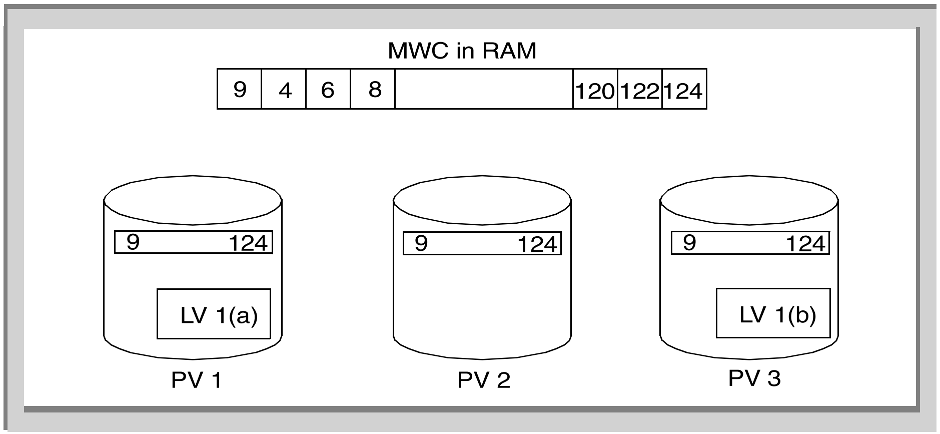


Figure 40. After the write to the mirrored logical volume

What happens if there is a crash?

Following is a detailed explanation of what happens if there is a crash during the write I/O.

Case (a): The crash occurs during the write of the MWC table in RAM. This is totally harmless. In this case, the real write to the disk, case (e) hasn't been started. The MWC table on the disk hasn't been altered. During reboot, the old MWC table will be used to re-synchronize the LTGs, writing the same data over again.

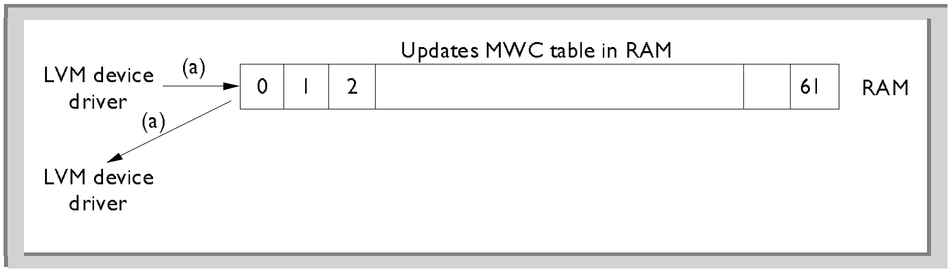


Figure 41. What happens if there is a crash? - Part 1

The solution is to issue parallel writes of the MWC table to each disk tracking the MWC (Note: This is only a write of 512 bytes of information to each disk).

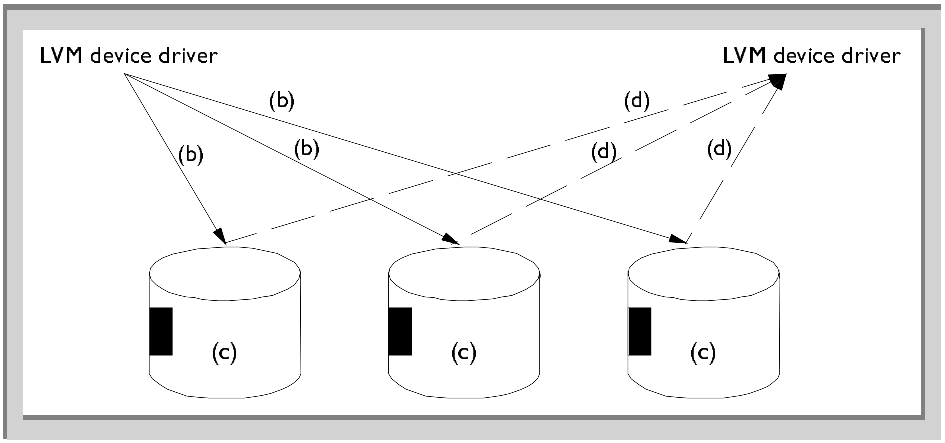


Figure 42. What happens if there is a crash? - Part 2

Case (b): The crash happens before the MWC table update to the disk platters can start: This case would be identical to case (a).

Case (c): The crash happens while the MWC table is being written to disk platters. This is harmless. In this case, the real write to the disk, case (e) hasn't been started. The MWC table may be inconsistent (all three copies of the MWC table don't match). However, this is a very low probability since the write was only 512 bytes. But, even if there is an inconsistent MWC, it doesn't matter. The inconsistent table will consist of 61 consistent entries and one inconsistent entry.

Case (d): The crash occurs after the MWC table has been updated. This is equivalent to case (c) except that the MWC entries on the disk platter will all be consistent. No LTG has actually changed.

At that point, the parallel MWC table writes all return back to the LVM device driver. We now have to write the real data. The size of this write is now between 512 bytes and 128 KB.

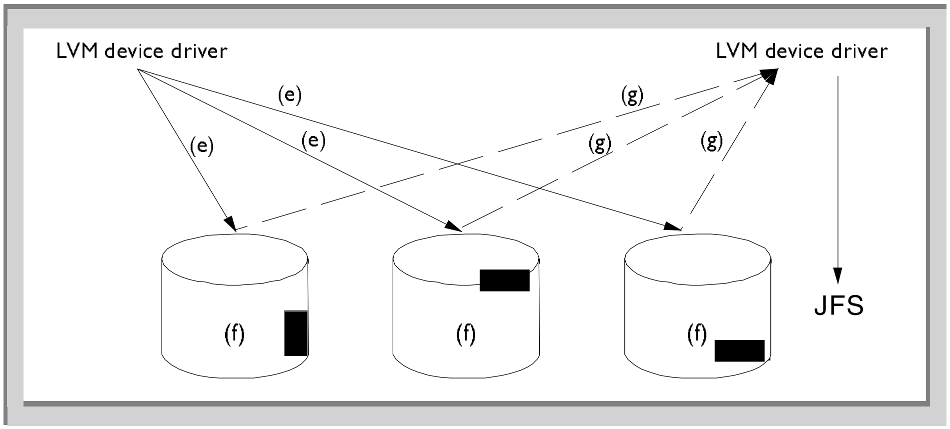


Figure 43. What happens if there is a crash? - Part 3

Case (e): The crash occurs before the LTG write can be issued to the disks. This is the same as case (d).

Case (f): The crash occurs during the LTG writes. This is the primary reason for the MWC to exist. In this case, you have three potentially large 128 KB writes that are occurring in parallel. There is no way to guarantee that, during the crash, any one of the mirrors have finished the write. And, if they didn't finish the write, they were in the write when the interruption occurred. It should be assumed that all three copies are in different stages of completion. Thus, upon reboot, the MWC is used as a guide to take one of the three copies as a master and copy over the two other mirror copies. If the one copy chosen is incomplete, it could theoretically overwrite another mirror copy that is actually complete! However, it is more important to guarantee *consistent* data than to have the most updated mirror write. Since the MWC table was written first, we know that the very last write issued to the disks is covered during re-synchronization.

Case (g): A crash occurs after LTG writes complete, but before the driver is notified of write completion. This is similar to case (f) except in this case there is no worry that the copy with the most new data is overwritten upon reboot. They all have completed their writes and any mirror chosen for the LTG re-synchronize will contain the latest data.

When all three writes complete and return back to the LVM device driver, then a *write complete* is signaled from LVM to the application.

Case (h): Crash before the LVM driver gets to tell JFS that the write completed. This is identical to case (g).

2.2.3.3 Ability to detect stale mirror copies and correct

The VGSA tracks the status of 1016 physical partitions per disk per volume group. During a read or write, if the LVM device driver detects that there was a failure in fulfilling a write request, the VGSA will note the physical partition(s) that failed and mark the partition(s) *stale*. When a partition is marked stale, this is logged by AIX error logging facility and the LVM device driver will be instructed not to send further partition data requests to that stale partition. This saves wasted time in sending I/O requests to a partition that most likely will not respond. And when this physical problem is corrected, the VGSA will tell the mirror synchronization code which partitions need to be updated to have the mirrors contain the same data.

If the operating system crashes in the middle of many writes, it has no time to mark a partition as stale. This is where MWC comes in; it assumes that the last 62 LTG writes were inconsistent and performs a sync, even though the partitions where the LTGs exist are not marked stale by the VGSA.

2.2.3.4 Data relocation

Relocation typically occurs when the system fails to perform a read or a write due to physical problems with the disk platter. In some cases, the data I/O request completes but with warnings. Depending on the type of recovered error, the LVM may order a relocation of the data to be on the safe side.

There are three types of data relocation:

- Physical level (internal to the disk)
- Intermediate level (hardware relocation ordered by LVM)
- Software level (software relocation)

The lowest layer of relocation is the one that is internal to the disk. These types of relocations are typically private to the disk and there is no notification to the user that a relocation occurred.

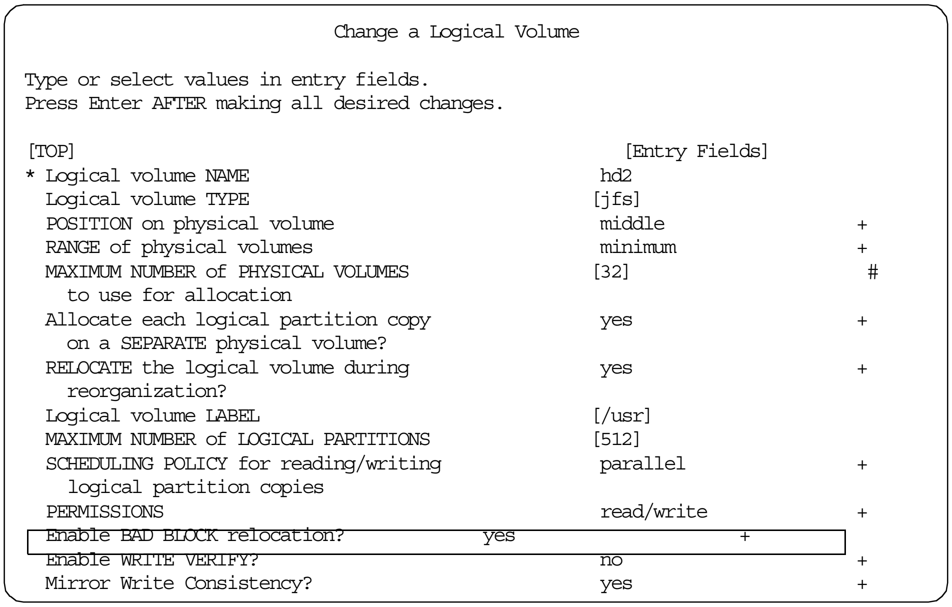
The next level up in terms of relocation complexity is an intermediate level (hardware relocation called for by the LVM device driver). This type of relocation is also called *bad block relocation* in the AIX LVM and will instruct the disk to relocate the data on one physical block to another block (reserved) of the disk. The location where the bad block is relocated is dependent on the disk itself. This bad block relocation information is held in the following area on the physical volume:

```
root@pukupuku:/ [101] # grep BB_ /usr/include/sys/hd_psn.h
#define  PSN_BB_DIR      8     /* beginning PSN of bad block directory  */
#define  LEN_BB_DIR      22    /* length in sectors of bad block dir    */
#define  PSN_BB_BAK      71    /* PSN of backup bad block directory     */
```

The top layer of data relocation is the software relocation handled by the LVM device driver. In this case, the LVM device driver maintains a bad block directory and whenever it receives a request to access a logical location A, the LVM device driver will look up the bad block table and translate it to actually send the request to the disk drive at physical location B. The default behavior of this bad block relocation function is determined by an attribute of the logical volume. To check this attribute, you can use following SMIT dialog.

```
# smit -C lv

    Set Characteristic of a Logical Volume
        Change a Logical Volume
```

Choose an appropriate logical volume name as LOGICAL VOLUME name.



```
                              Change a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                    [Entry Fields]
* Logical volume NAME                                  hd2
  Logical volume TYPE                                 [jfs]
  POSITION on physical volume                          middle              +
  RANGE of physical volumes                            minimum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  [32]                  #
    to use for allocation
  Allocate each logical partition copy                 yes                 +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                   yes                 +
    reorganization?
  Logical volume LABEL                                [/usr]
  MAXIMUM NUMBER of LOGICAL PARTITIONS                [512]
  SCHEDULING POLICY for reading/writing                parallel            +
    logical partition copies
  PERMISSIONS                                          read/write          +
  Enable BAD BLOCK relocation?            yes                   +
  Enable WRITE VERIFY?                                 no                  +
  Mirror Write Consistency?                            yes                 +
```

Otherwise, you can examine this from the command line:

```
root@pukupuku:/ [132] # lslv hd2
LOGICAL VOLUME:     hd2                    VOLUME GROUP:   rootvg
LV IDENTIFIER:      00702137e70a7982.5     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                152                    PPs:            152
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        /usr                   LABEL:          /usr
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
```

By default, AIX will not allow the execution of the intermediate-level data relocation on non-IBM drives. On IBM drives, the AIX LVM relies on the guarantee that there are reserved areas for possible hardware relocation (typically 256 blocks per disk). However, on non-IBM drives, AIX LVM cannot predict what spare blocks may or may not exist for possible relocation. So, assuming the worst, AIX LVM only allows the internal disk relocation and the top layer software relocation to occur on non-IBM drives.

The non-IBM disk discussed previously also include the drive manufactured by IBM but not designed for RS/6000 and AIX. Usually, these disks are classified as osdisk in AIX:

```
root@pukupuku:/ [288] # lsdev -Cc disk -t osdisk
hdisk5 Available 04-01-00-0,0 Other SCSI Disk Drive
root@pukupuku:/ [289] # lscfg -vl hdisk5
  DEVICE            LOCATION          DESCRIPTION

  hdisk1            04-01-00-1,0      Other SCSI Disk Drive

        Manufacturer................IBM
        Machine Type and Model......DDRS-39130W
        Part Number.................Corp. 1997.
        ROS Level and ID...........53393742
        Serial Number...............REFD7615
        EC Level....................All rights
        FRU Number.................. reserved.
        Device Specific.(Z0)........000002029F00003A
        Device Specific.(Z1)........
        Device Specific.(Z2)........) Co
        Device Specific.(Z3)........pyrig
        Device Specific.(Z4)........t IB
        Device Specific.(Z5)........M
        Device Specific.(Z6)........
```

The above example shows that some VPD (Vital Product Data) attributes for this drive cannot be parsed (or accepted) correctly (for example, you can see it on the Data Specific lines from Z0 to Z4, where it should read Copyright IBM) by the device driver during the SCSI probe phase, thus this drive is classified as osdisk.

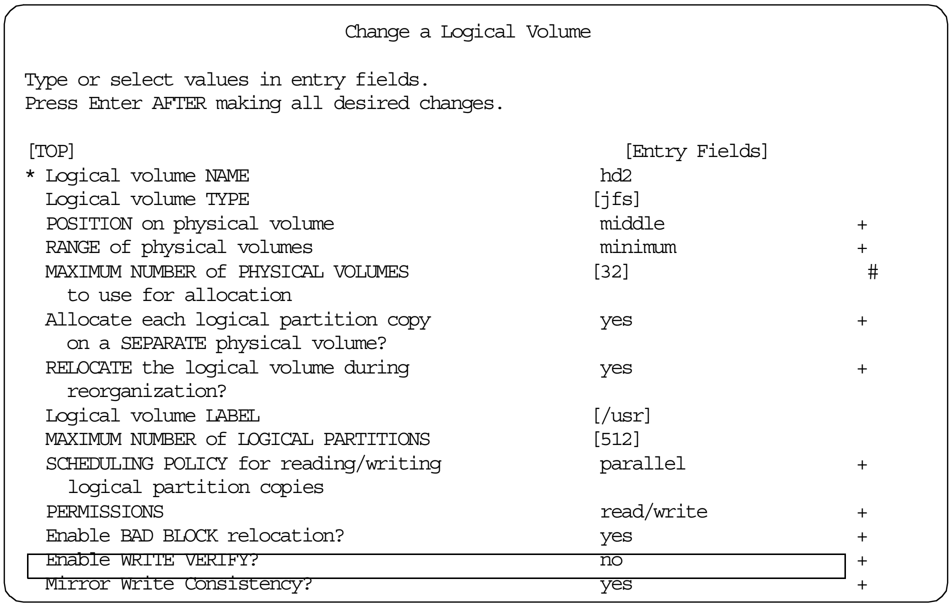
2.2.3.5 Write verify

The AIX LVM provides the possibility to specify that you wish an extra level of data integrity to be achieved every time you write data to the disk. This is the ability known as *write verify*. This capability is given to each logical volume in a volume group.

Actually, the default behavior of this write verify function is determined by an attribute of the logical volume. To check this attribute, you can use the following SMIT dialog.

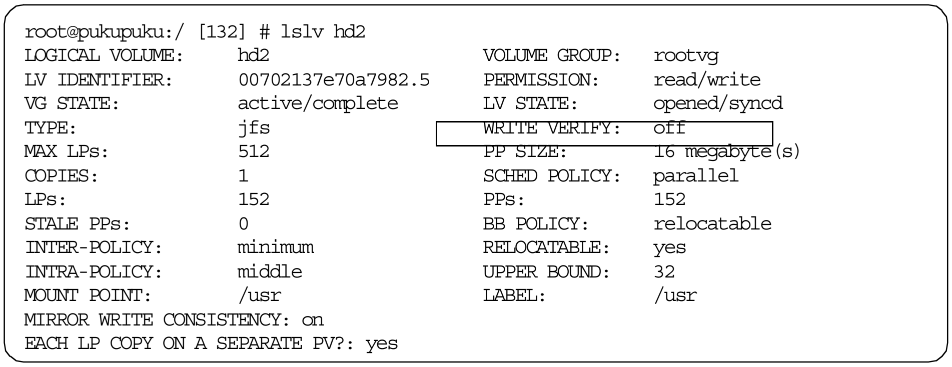
```
# smit -C lv

   Set Characteristic of a Logical Volume
       Change a Logical Volume
```

Choose the appropriate logical volume name as LOGICAL VOLUME name.



```
                          Change a Logical Volume

 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

 [TOP]                                               [Entry Fields]
 * Logical volume NAME                             hd2
   Logical volume TYPE                            [jfs]
   POSITION on physical volume                     middle            +
   RANGE of physical volumes                       minimum           +
   MAXIMUM NUMBER of PHYSICAL VOLUMES             [32]                #
     to use for allocation
   Allocate each logical partition copy            yes               +
     on a SEPARATE physical volume?
   RELOCATE the logical volume during              yes               +
     reorganization?
   Logical volume LABEL                           [/usr]
   MAXIMUM NUMBER of LOGICAL PARTITIONS           [512]
   SCHEDULING POLICY for reading/writing           parallel          +
     logical partition copies
   PERMISSIONS                                     read/write        +
   Enable BAD BLOCK relocation?                    yes               +
   Enable WRITE VERIFY?                            no                +
   Mirror Write Consistency?                       yes               +
```

Otherwise, you can examine this from the command line:



```
root@pukupuku:/ [132] # lslv hd2
LOGICAL VOLUME:     hd2                    VOLUME GROUP:   rootvg
LV IDENTIFIER:      00702137e70a7982.5     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                152                    PPs:            152
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        /usr                   LABEL:          /usr
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
```

When you have the write verify attribute enabled, every write to a physical portion of a disk that is part of a logical volume causes the disk device driver to issue the `Write` and `Verify SCSI` command to the disk. This means that after each write, the disk will reread the data and do an IOCC (I/O Channel Controller; SCSI command protocol) parity check on the data to see if what the platter wrote exactly matched what the write request buffer contained. This type of extra check understandably adds more time to the completion length of a write request, but it adds to the integrity of the system.

2.3 Practical examples

This section shows real examples of managing the mirrored logical volumes.

2.3.1 Example configurations

All the examples shown in this section are examined with the following configurations:

```
root@itsosrv1:/ [104] # lsdev -Cc disk
(many lines are snipped out)
hdisk6  Available 30-60-L       SSA Logical Disk Drive
hdisk9  Available 30-60-L       SSA Logical Disk Drive
root@itsosrv1:/ [107] # mkvg -s 8 -y mirrorvg hdisk6 hdisk9
root@itsosrv1:/ [108] # lspv
hdisk6         00017d37914ed685    mirrorvg
hdisk9         00017d37628265fc    mirrorvg
```

2.3.1.1 Create copies of pre-existing logical volumes

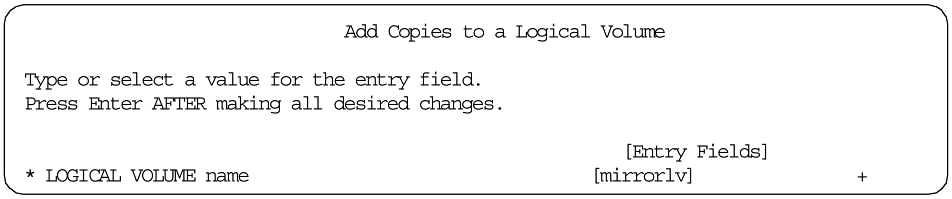
Suppose the logical volume already exists on one physical volume.

```
root@itsosrv1:/ [109] # mklv -y mirrorlv mirrorvg 10 hdisk6
mirrorlv
root@itsosrv1:/ [110] # lsvg -l mirrorvg
mirrorvg:
LV NAME            TYPE       LPs    PPs   PVs  LV STATE      MOUNT POINT
mirrorlv           jfs        10     10    1    closed/syncd  N/A
```

The following example shows how to create a copy of this logical volume using SMIT.

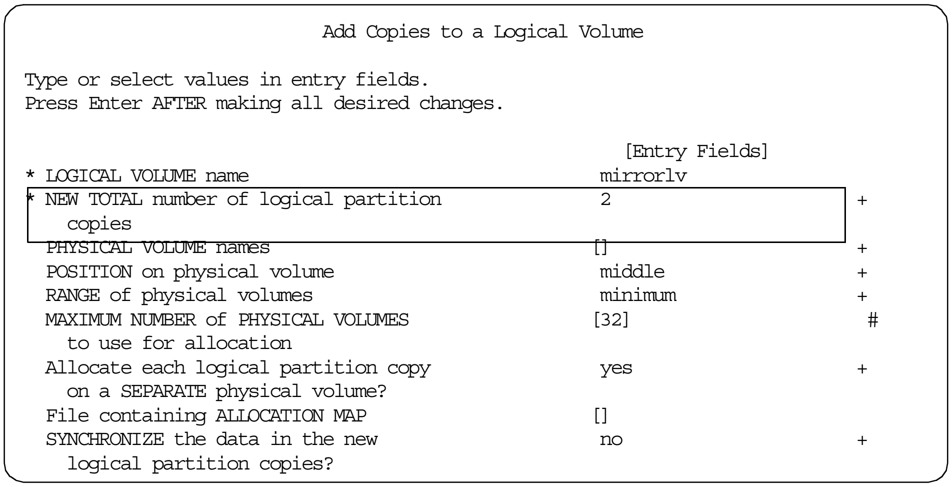
```
root@itsosrv1:/ [111] # smit -C lv
    Set Characteristic of a Logical Volume
         Add a Copy to a Logical Volume
```

Enter the logical volume name as LOGICAL VOLUME name. Then press **Enter**.



```
                              Add Copies to a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* LOGICAL VOLUME name                                [mirrorlv]            +
```

Press the **F4** key on the NEW TOTAL number of logical partition copies line, then choose the copy number (1, 2, 3). In this case, we choose 2 as the copy number. Then press **Enter**.



```
                            Add Copies to a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* LOGICAL VOLUME name                                 mirrorlv
* NEW TOTAL number of logical partition               2                     +
    copies
  PHYSICAL VOLUME names                              []                     +
  POSITION on physical volume                         middle                +
  RANGE of physical volumes                           minimum               +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                 [32]                    #
    to use for allocation
  Allocate each logical partition copy                yes                   +
    on a SEPARATE physical volume?
  File containing ALLOCATION MAP                     []
  SYNCHRONIZE the data in the new                     no                    +
    logical partition copies?
```

If you prefer the command line interface, you can use the following command line:

```
root@itsosrv1:/ [116] # mklvcopy mirrorlv 2
```

As a result, you will have the following output:

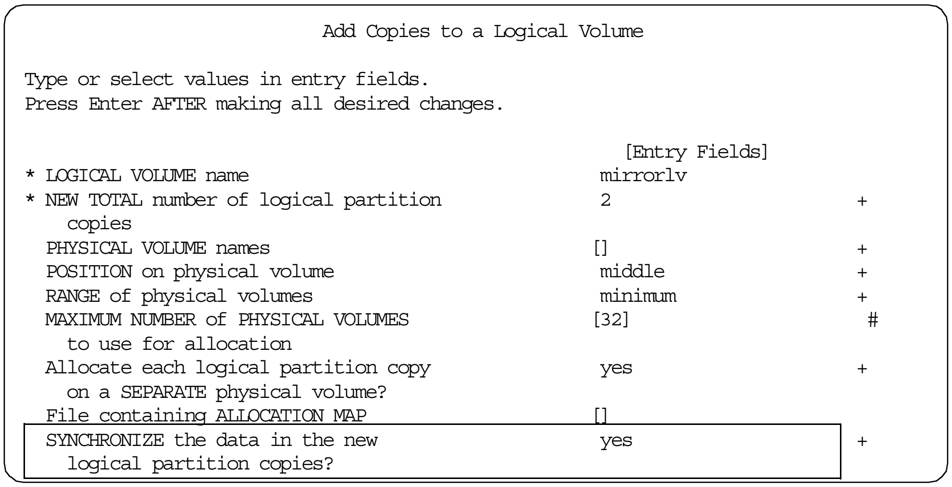
```
root@itsosrv1:/ [114] # lsvg -l mirrorvg
```

```
mirrorvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
mirrorlv            jfs        10    20    2    closed/stale  N/A
```

Note that the physical partitions are just allocated, and the data on this logical volume is not synchronized among the copies at this point (therefore, it is marked as stale). To synchronize the staled mirrored logical volumes, you have to issue the following command:

```
root@itsosrv1:/ [117] # syncvg -l mirrorlv
root@itsosrv1:/ [118] # lsvg -l mirrorvg
mirrorvg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
mirrorlv            jfs        10    20    2    closed/synced N/A
```

If you want to synchronize the copies at the creation time, then you have to set the ‘SYNCHRONIZE the data in the new logical partition copies’ message to yes.



```
                           Add Copies to a Logical Volume

 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
 * LOGICAL VOLUME name                                  mirrorlv
 * NEW TOTAL number of logical partition                2                      +
     copies
   PHYSICAL VOLUME names                               []                      +
   POSITION on physical volume                          middle                 +
   RANGE of physical volumes                            minimum                +
   MAXIMUM NUMBER of PHYSICAL VOLUMES                  [32]                     #
     to use for allocation
   Allocate each logical partition copy                 yes                    +
     on a SEPARATE physical volume?
   File containing ALLOCATION MAP                      []
   SYNCHRONIZE the data in the new                      yes                    +
     logical partition copies?
```

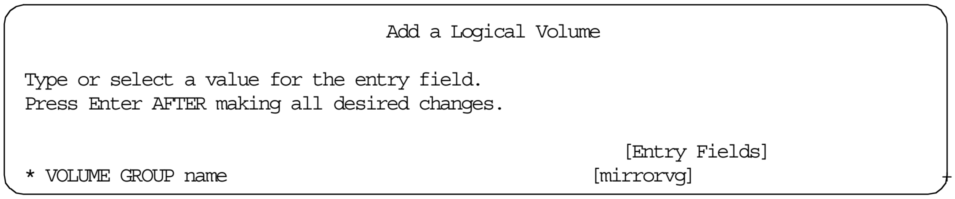
If you attempt to synchronize huge logical volumes, this operation will take some time. During hat operation, any update operation of the LVM metadata (VGDA, VGSA and so on) in this volume group is prohibited. To minimize the synchronization time, it is recommended to create the copies of the logical volumes first, then to issue the `syncvg` or the `mirrorvg` command to synchronize all the copies at one time.

2.3.1.2 Create copies during the creation of the logical volume

You can create several copies of a logical volumes during its initial creation. The following example shows how to create the copy of the logical volume using SMIT.

```
root@itsosrv1:/ [117] # smit -C lv
   Add a Logical Volume
```

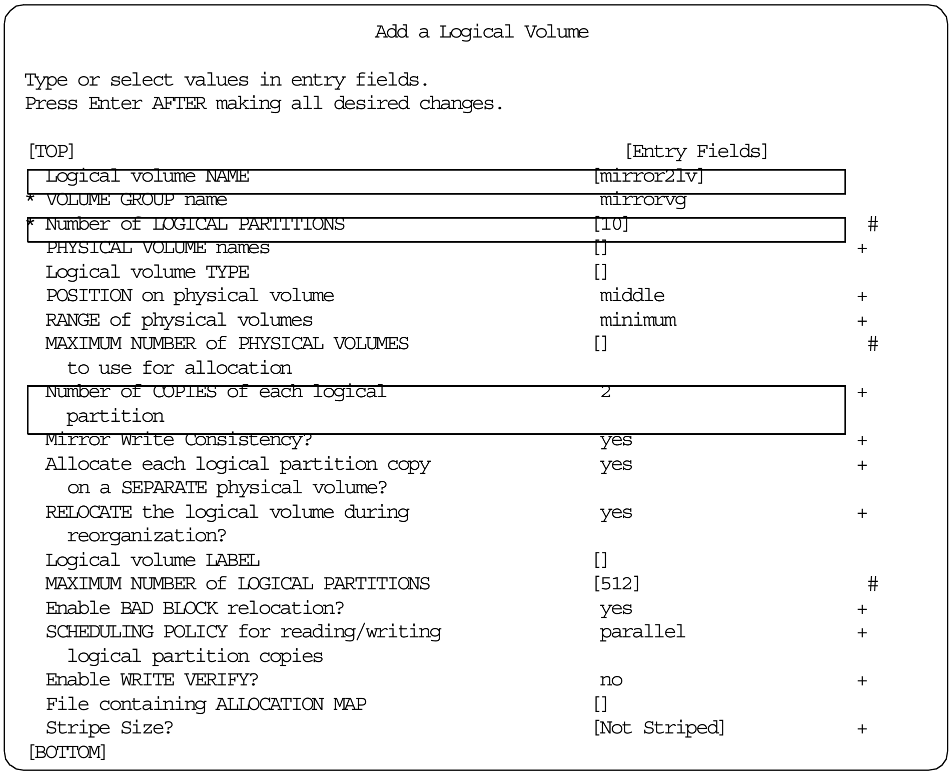
Enter the volume group name and press **Enter**.



```
                               Add a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                             [Entry Fields]
* VOLUME GROUP name                                        [mirrorvg]                  +
```

Enter the following necessary information and press **Enter**:

- Logical volume NAME
- Number of LOGICAL PARTITIONS
- Number of COPIES of each logical partition



```
                                 Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                    [Entry Fields]
  Logical volume NAME                                   [mirror2lv]
* VOLUME GROUP name                                      mirrorvg
* Number of LOGICAL PARTITIONS                          [10]              #
  PHYSICAL VOLUME names                                 []               +
  Logical volume TYPE                                   []
  POSITION on physical volume                            middle          +
  RANGE of physical volumes                              minimum         +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                    []                #
    to use for allocation
  Number of COPIES of each logical                       2               +
    partition
  Mirror Write Consistency?                              yes             +
  Allocate each logical partition copy                   yes             +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                     yes             +
    reorganization?
  Logical volume LABEL                                  []
  MAXIMUM NUMBER of LOGICAL PARTITIONS                  [512]             #
  Enable BAD BLOCK relocation?                           yes             +
  SCHEDULING POLICY for reading/writing                  parallel        +
    logical partition copies
  Enable WRITE VERIFY?                                   no              +
  File containing ALLOCATION MAP                        []
  Stripe Size?                                          [Not Striped]    +
[BOTTOM]
```

If you prefer the command line interface, you can use the following command line:

```
root@itsosrv1:/ [119] # mklv -y mirror2lv -c 2 mirrorvg 10
```

As a result, you will have the following output:

```
root@itsosrv1:/ [119] # lsvg -l mirrorvg
mirrorvg:
LV NAME          TYPE      LPs   PPs   PVs  LV STATE      MOUNT POINT
mirrorlv         jfs       10    20    2    closed/stale  N/A
mirror2lv        jfs       10    20    2    closed/syncd  N/A
```

2.4 Mirroring of the rootvg

This section will attempt to show in detail why the rootvg mirroring has such special treatment in the AIX system management.

2.4.1 Brief explanation about the AIX boot sequence

Before describing the AIX boot sequence, several key definitions are given here:

blv It is the abbreviation for *boot logical volume* also known as hd5. It contains the minimal file system needed to begin the boot of a system (This file system is held in compressed format, and only a limited number of commands, like `savebase` and `bosboot` can manipulate it). An important portion of this minimal file system is the mini-ODM.

bootrec It is the first disk block on the boot disk that the Initial Program Load Read Only Storage (IPL ROS) reads to find the reference to the blv. It is defined in /usr/include/sys/hd_psn.h as follows:

```
root@pukupuku:/ [320] # grep IPL /usr/include/sys/hd_psn.h
#define  PSN_IPL_REC     0     /* PSN of the IPL record                 */
```

`savebase` This command will update the mini-ODM that resides on the same disk as /dev/ipldevice.

The relationship between the bootrec, the blv, and the mini-ODM is illustrated in Figure 44.

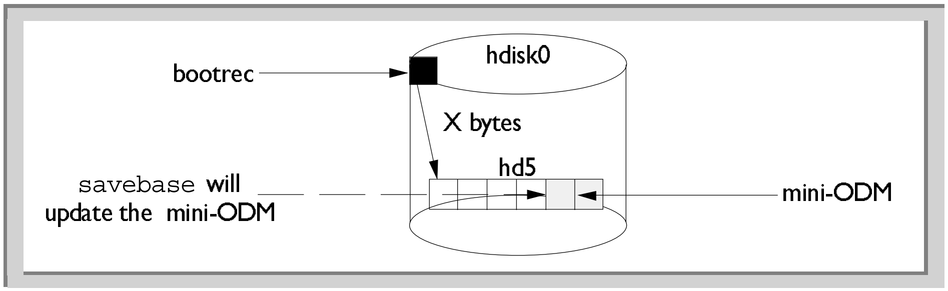


Figure 44. Three important components for booting rootvg

The bootrec is read by the IPL ROS code, it tells the ROS that it needs to jump X bytes into the disk platter to read the boot logical volume, hd5.

Important

This is the first reason why the rootvg mirroring requires special treatment. The bootrec is not synchronized by AIX LVM, since it is not a logical volume.

The processor starts reading in the mini file system to start the AIX system boot. Note that this is a very simplified view of the boot process, but enough for a basic understanding of the upcoming conflicts.

During the processing of the mini-AIX file system, there is a mini-ODM read into the RAM. When the real rootvg file system comes online, AIX merges the data held in the mini-ODM with the real ODM held in /etc/objrepos in the root file system.

Whenever an LVM command is executed that will change the mini-ODM, there is a command called `savebase` that is run at the successful completion of the LVM command (represented as a) in Figure 44 on page 133). The `savebase` takes a snapshot of the main ODM and compresses it and picks out only what is needed for boot, such as LVM information concerning logical volumes in rootvg, and so on. It takes this compressed boot image and looks at /dev/ipldevice and finds out what disk this represents. Typically on most systems, /dev/ipldevice is a hard link to /dev/hdisk0 (which contains the rootvg volume group):

```
root@pukupuku:/ [323] # ls -l /dev/ipldevice /dev/*hdisk0
brw-------   1 root     system    15,  1 Sep 03 07:50 /dev/hdisk0
crw-------   2 root     system    15,  1 Apr 24 18:38 /dev/ipldevice
crw-------   2 root     system    15,  1 Apr 24 18:38 /dev/rhdisk0
```

The /dev/ipldevice is really needed to indicates which disk holds hd5.

2.4.2 Contiguity of the boot logical volume

As described in earlier, the blv (boot logical volume or hd5) is read by the IPL ROS code directly during the boot phase. The IPL ROS code will completely bypass the LVM access routines, and access the memory in sequence, assuming that all this code is stored in a unique place. With the increase of the IPL code, it may happen that one partition is not enough. If you add a new partition to the blv, you must make sure that the second partition will be next to the first one, the IPL code won’t be able to jump from partition 1 to partition 10, for example.

To confirm that the partitions of the blv are contiguous, you have to use following command:

```
root@pukupuku:/ [128] # lslv -m hd5
hd5:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0001 hdisk0
```

In our case, the blv is made of one logical partition and resides in the first physical partition on hdisk0. If this output shows multiple lines (this means that the blv was composed of multiple logical partitions), you may have to organize it to occupy contiguous regions.

2.4.3 Dump device

The dump device is a raw logical volume (usually having sysdump as its logical volume type) for the sake of the AIX kernel writing its memory image (also known as core image) upon system crash phase.

In AIX, the dump device should reside in rootvg, and it is set to /dev/hd6 by default. The /dev/hd6 is also the initial paging device. After the system has crashed, during the reboot phase, the boot procedure will copy this dump image from the dump device to the /var/adm/ras directory as a file named vmcore to prevent the VMM (virtual memory manager) from overwriting its space. Further details about the system dump, please consult the *AIX Version 4.3 Problem Solving Guide and Reference*, SC23-4123.

In this section, the following assumptions are made (actually, these assumptions are reasonable from a system management perspective):

- The rootvg is mirrored using two physical volumes (hdisk0, hdisk1) which have same size.
- You have made /dev/hd7 a primary dump device on hdisk0, and /dev/sysdumpnull the second dump device using following command:

```
# sysdumpdev -p /dev/hd7 -P
# sysdumpdev -s /dev/sysdumpnull -P
# sysdumpdev -K (it enables the user initiated system dump)
# shutdown -Fr (to take effects above change)
```

Then, you will see the following system dump configuration:

```
root@pukupuku:/ [336] # sysdumpdev -l
primary              /dev/hd7
secondary            /dev/sysdumpnull
copy directory       /var/adm/ras
forced copy flag     TRUE
always allow dump    TRUE
dump compression     OFF
```

Before AIX Version 4.3.3, there is no complete support for the mirrored dump device. Since, if the system dumped, the dump image was written to the first mirror copy only without any staleness, bypassing the LVM device driver. The dump image is certainly written in the first mirror copy, but when it was read, LVM simply started to pass back randomly-selected mirror copies for each logical partition, so the data came back in a corrupted state.

Actually, these work-arounds are performed by the `mirrorvg` command automatically after AIX Version 4.2.1 (before this release, there was no `mirrorvg` command provided. Thus, you had to follow the official step-by-step method shown in Appendix A, “Mirroring of the rootvg” on page 353).

With AIX Version 4.3.3, you can use the `readlvcopy` command to read an individual mirror copy of any logical volumes. Hence, `readlvcopy` is now used in the `snap` command to copy the dump data from the first mirror copy only, and thus obtain the original dump data without corruption.

Chapter 3. Striping

This chapter describes the striping function included in the AIX Logical Volume Manager. We will see the concepts and study real examples.The mirroring and striping function, introduced in AIX Version 4.3.3, is also described. The performance issues of using striped logical volumes are covered in Chapter 7, “Performance” on page 303.

3.1 Concept

The striping mechanism, also known as RAID 0, is a technology that was developed to achieve I/O performance gain. The basic concept of striping is that in the write phase, each data is chopped into small pieces (called striped unit, or chunk) and these chunks are written to separate physical volumes in parallel. In the read phase, these chunks are read from those separate physical volumes in parallel, and re-assembled into the actual data. From the high-availability point of the view, the striping function does not provide redundancy by any means (except for the mirroring and striping function introduced by AIX Version 4.3.3, refer to 3.3, “The mirror and stripe function” on page 162). The striping function support was introduced in AIX Version 4.1.

This section describes the concept for the striping function and covers the necessary information for managing striped logical volumes in AIX.

3.1.1 Basic schema of the striped logical volumes

This section attempts to provide basic schema for the striped logical volumes using graphical explanations. In Figure 45, you have a volume group (stripevg) that is composed of three physical volumes having the same size (PV1, PV2, and PV3). A striped logical volume (stripelv) is created on this volume group, which spreads over the three physical volumes.

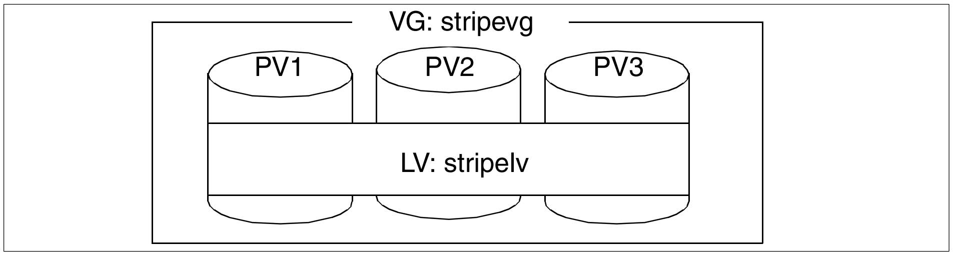


Figure 45. The striped logical volume outline

3.1.1.1 The logical partitions mapping scheme

In the striped logical volumes, all the logical partitions are mapped to physical partitions on these physical volumes in a round-robin fashion (see Figure 46). In this example, the logical partition 2 is mapped to physical partition 1 on the first physical volume (PV1), and the logical partition 2 is mapped to the physical partition 1 on the second physical volume (PV2), and so on. In other words, the physical partitions are equally allocated on each physical volume.

This logical partitions to physical partitions mapping scheme is achieved by the inter-physical volume allocation set to maximum. For further information about the inter-physical volume allocation, refer to 3.1.2, “Inter physical volume allocation policy” on page 141.

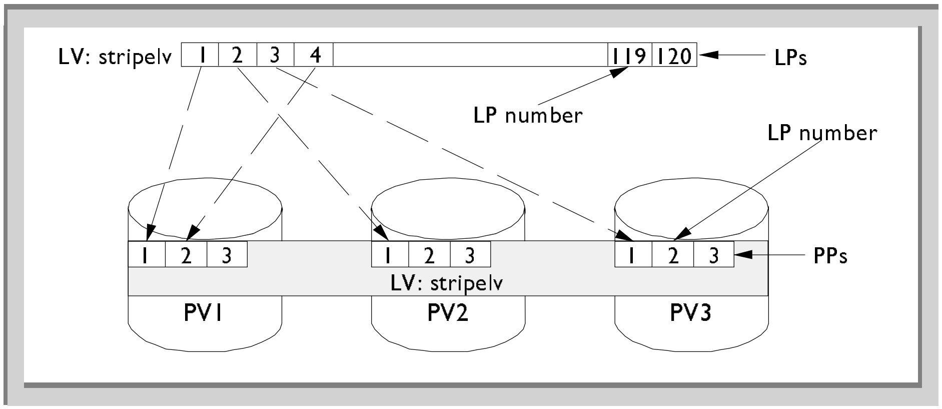


Figure 46. The striped logical volume - Physical partition mapping

3.1.1.2 The physical partitions are divided into many chunks

When your applications access the striped logical volumes, the storage area for each physical partition is not used contiguously. These physical partitions are divided into chunks. The chunks size may be 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, or 128 KB. This size is determined at the creation of the striped logical volumes, and cannot be changed after.

The chunk size is also called *stripe unit size* or *stripe length*. The number of the physical volumes that accommodate the striped logical volume is also called *stripe width*.

In Figure 47, each physical partition is divided into chunks. Each of them is represented as X-Y, where X is the physical partition number, and Y is the chunk occurrence number in the physical partition.

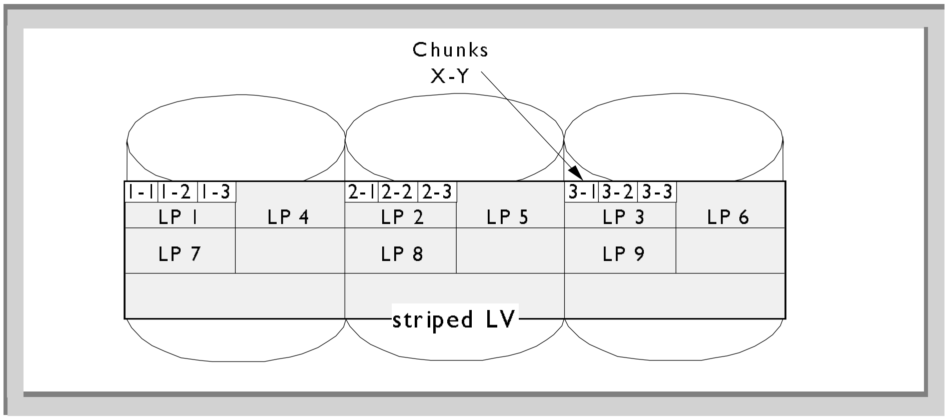


Figure 47. Physical partitions are divided into many chunks

If your applications access this striped logical volume, then the AIX LVM supplies the chunks in the following order: 1-1, 2-1, 3-1, 1-2, and so on, in a round-robin fashion.

Note

In the striped logical volumes, not only the allocation of the physical partitions, but also the chunks, are accessed on each physical volume in this round-robin fashion.

3.1.1.3 Write phase

In the write phase, a caller (like the journaled file system device driver) issues one write I/O request to the AIX LVM device driver (step 1 in Figure 48). This request is chopped into several chunks (step 2), and they are written to each physical disk in parallel (step 3).

Since these three writes (to the chunks 1-1, 2-1, and 3-1) are achieved on separate drives, these writes are executed in parallel. It improves the write performance compared to one drive. Subsequently, if all the writes to each physical volume return with no error (step 4), then the LVM device driver returns a success to the caller (step 5). Otherwise, it returns an error to the caller.

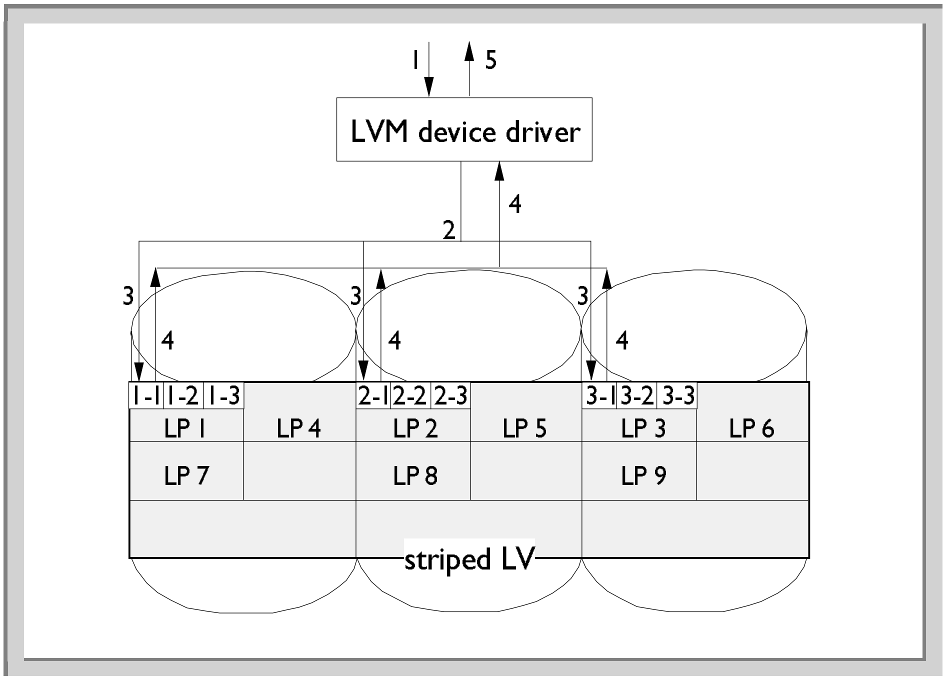


Figure 48. The striped logical volume - Write phase behavior

3.1.1.4 Read phase

In the read phase, a caller (like the journal file system device driver) issues one read I/O request to the AIX LVM device driver (step 1 in Figure 49 on page 141). This request is split into several read calls for chunks (step 2), and these small reads are sent to each physical disk in parallel. Since these three reads (from the chunks 1-1, 2-1, and 3-1) are achieved on separate drives, these reads are executed in parallel (step 3). It improves the read performance compared to one physical drive.

Subsequently, if all the reads to each physical volume return with no error (step 4), the LVM device driver directly passes the memory pointers for all the chunks that compose the data into the user's memory area without re-assembly (step 5). If one of the read calls for chunks failed, then the AIX LVM device driver would return an error to the caller.

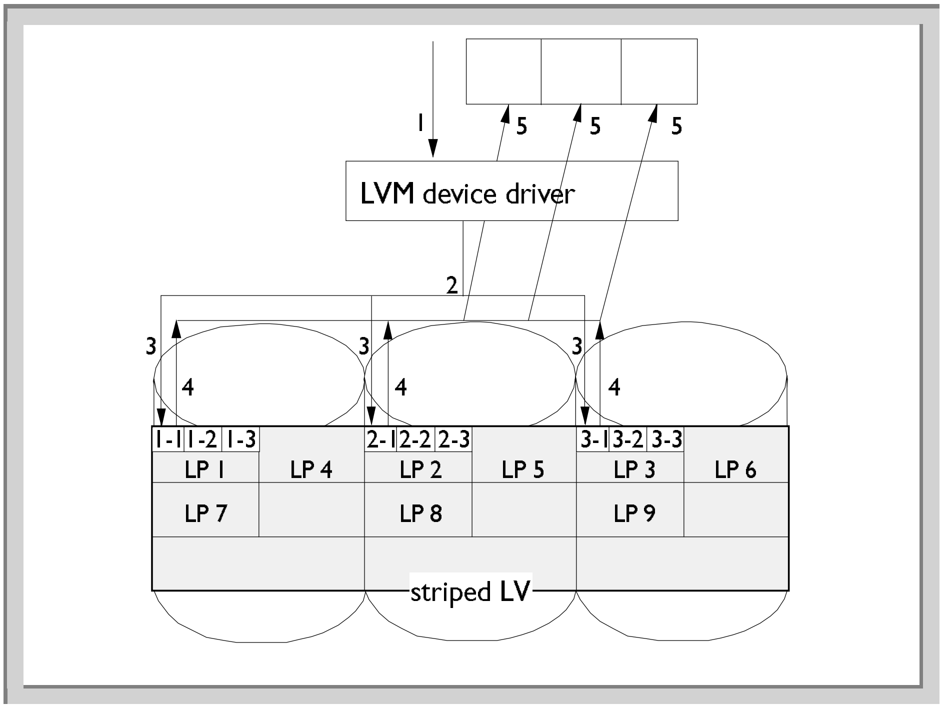


Figure 49. The striped logical volume: The read phase

Note

The AIX LVM does not re-assemble these chunks into one read request. The chunks are read from each physical volume, then write into the caller's memory area directly.

3.1.2 Inter physical volume allocation policy

The inter-physical volume allocation policy is one of the attributes of the logical volumes. It stipulates how many physical volumes are used for the allocation of the physical partitions for the logical volume. Usually (meaning non-striped logical volume case) it is set to *m* (minimum) by default to minimize the physical volumes that are used for the allocation of the physical partitions of the logical volume. It is adequate, since if one of the physical volumes that accommodates this logical volume fails, then the entire logical volume will be lost.

To confirm the inter-physical volume allocation policy of the logical volume, you can use the `lslv` command as shown:

```
root@lorraine:/ [113] # lslv hd6
LOGICAL VOLUME:     hd6                    VOLUME GROUP:   rootvg
LV IDENTIFIER:      00041631ae592047.2     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               paging                 WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                64                     PPs:            64
STALE PPs:          0                      BB POLICY:      non-relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV?: yes
```

In this example, the inter-physical volume allocation policy is set to minimum (shown as minimum in the rectangle). Since this is not a striped logical volume, there is no reason to set the inter-physical volume allocation policy to maximum. It is the default value for non-striped logical volumes.

3.1.2.1 How does the inter-physical volume allocation policy work?

In most circumstances, there is no need to set the inter-physical volume allocation policy set to maximum (x). However, you might want to know how it works. The answer is described in the following sections.

Example of configurations

All the examples shown in this section are examined with the following configuration:

- AIX Version 4.3.3
- RS/6000 model F50
- PPC604e 332MHz x 4 ways
- 512MB RAM
- 2 x 9.1GB SCSI2 F/W disk (hdisk0 and hdisk3)
- 4 x 4.5GB Ultra SCSI F/W disk (the other LVD connection disks)

Here is the listing of the disks on our system:

```
root@lorraine:/ [121] # lsdev -Cc disk
hdisk0 Available 10-60-00-8,0  16 Bit SCSI Disk Drive
hdisk1 Available 10-60-00-9,0  16 Bit LVD SCSI Disk Drive
hdisk2 Available 10-60-00-10,0 16 Bit LVD SCSI Disk Drive
hdisk3 Available 30-58-00-8,0  16 Bit SCSI Disk Drive
hdisk4 Available 30-58-00-9,0  16 Bit LVD SCSI Disk Drive
```

```
hdisk5 Available 30-58-00-10,0 16 Bit LVD SCSI Disk Drive
```

Here is the creation of the volume group, which will contain the striped logical volumes:

```
root@lorraine:/ [122] # mkvg -y pmstripevg -s 8 -t 2 hdisk1
0516-1193 mkvg: WARNING, once this operation is completed, volume group pmstripevg
        cannot be imported into AIX 430 or lower versions. Continue (y/n)?
y
pmstripevg
root@lorraine:/ [137] # lsvg pmstripevg
VOLUME GROUP:   pmstripevg               VG IDENTIFIER:  00041631540cc083
VG STATE:       active                   PP SIZE:        8 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      537 (4296 megabytes)
MAX LVs:        256                      FREE PPs:       537 (4296 megabytes)
LVs:            0                        USED PPs:       0 (0 megabytes)
OPEN LVs:       0                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     1                        AUTO ON:        yes
MAX PPs per PV: 2032                     MAX PVs:        16
root@lorraine:/ [129] # extendvg -f pmstripevg hdisk2
root@lorraine:/ [130] # extendvg -f pmstripevg hdisk4
root@lorraine:/ [131] # extendvg -f pmstripevg hdisk5
root@lorraine:/ [132] # lsvg
pmstripevg
rootvg
```

Here is the resulting volume group:

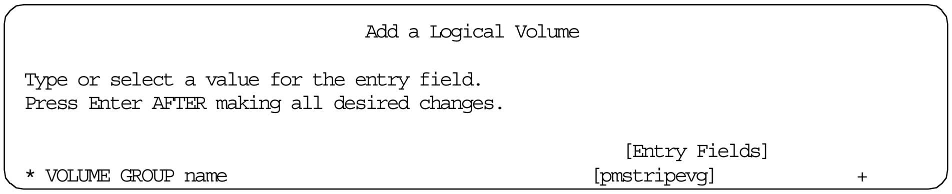
```
root@lorraine:/ [133] # lsvg pmstripevg
VOLUME GROUP:   pmstripevg               VG IDENTIFIER:  00041631540cc083
VG STATE:       active                   PP SIZE:        8 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      2148 (17184 megabytes)
MAX LVs:        256                      FREE PPs:       2148 (17184 megabytes)
LVs:            0                        USED PPs:       0 (0 megabytes)
OPEN LVs:       0                        QUORUM:         3
TOTAL PVs:      4                        VG DESCRIPTORS: 4
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     4                        AUTO ON:        yes
MAX PPs per PV: 2032                     MAX PVs:        16
```

3.1.2.2 The poor man's stripe

To create the logical volume with the inter-physical volume allocation policy set to maximum (x), you can use following SMIT panel.

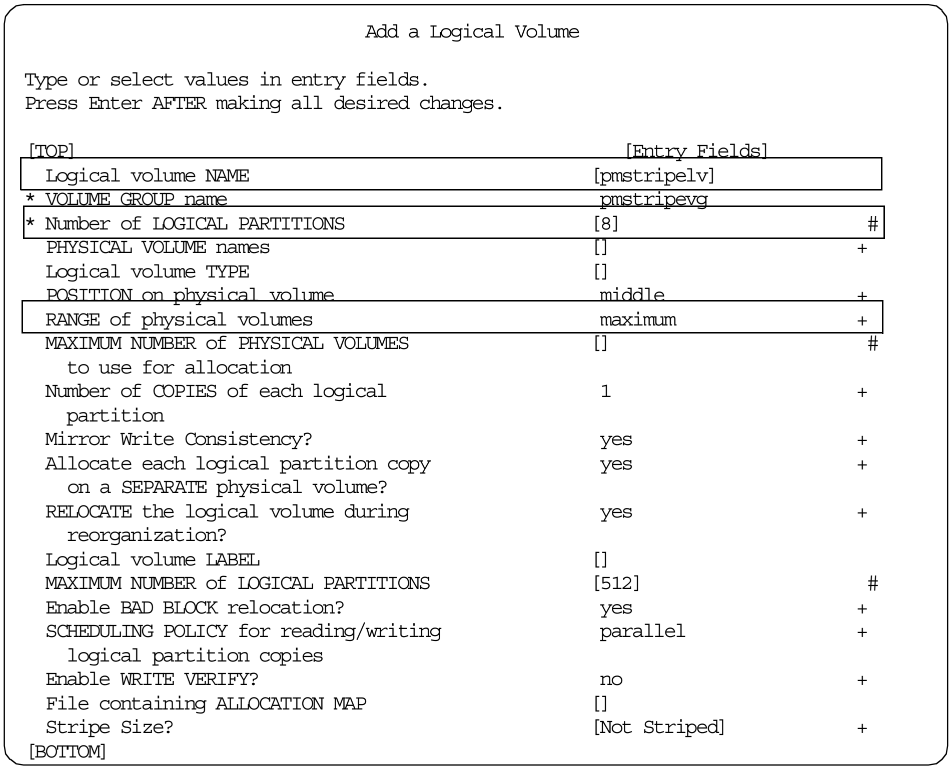
```
root@lorraine:/ [141] # smit lv
    Add a Logical Volume
```

Enter the volume group name in the following dialog.



```
                                Add a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                     [Entry Fields]
* VOLUME GROUP name                                [pmstripevg]           +
```

Now enter the necessary information. In this example, we specified the information shown in the rectangles in the following SMIT dialog panel screen.



```
                              Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
  Logical volume NAME                                  [pmstripelv]
* VOLUME GROUP name                                     pmstripevg
* Number of LOGICAL PARTITIONS                         [8]                  #
  PHYSICAL VOLUME names                                []                  +
  Logical volume TYPE                                  []
  POSITION on physical volume                           middle             +
  RANGE of physical volumes                             maximum            +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                   []                   #
    to use for allocation
  Number of COPIES of each logical                      1                  +
    partition
  Mirror Write Consistency?                             yes                +
  Allocate each logical partition copy                  yes                +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                    yes                +
    reorganization?
  Logical volume LABEL                                 []
  MAXIMUM NUMBER of LOGICAL PARTITIONS                 [512]                #
  Enable BAD BLOCK relocation?                          yes                +
  SCHEDULING POLICY for reading/writing                 parallel           +
    logical partition copies
  Enable WRITE VERIFY?                                  no                 +
  File containing ALLOCATION MAP                       []
  Stripe Size?                                         [Not Striped]       +
[BOTTOM]
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [146] # mklv -y pmstripelv -e x pmstripevg 8
```

To examine how the physical partitions are allocated, we use the `lslv` command:

```
root@lorraine:/ [147] # lslv -m pmstripelv
pmstripelv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0109 hdisk1
0002  0109 hdisk2
0003  0109 hdisk4
0004  0109 hdisk5
0005  0110 hdisk1
0006  0110 hdisk2
0007  0110 hdisk4
0008  0110 hdisk5
```

Figure 50 on page 145 shows the same information in a graphical output.

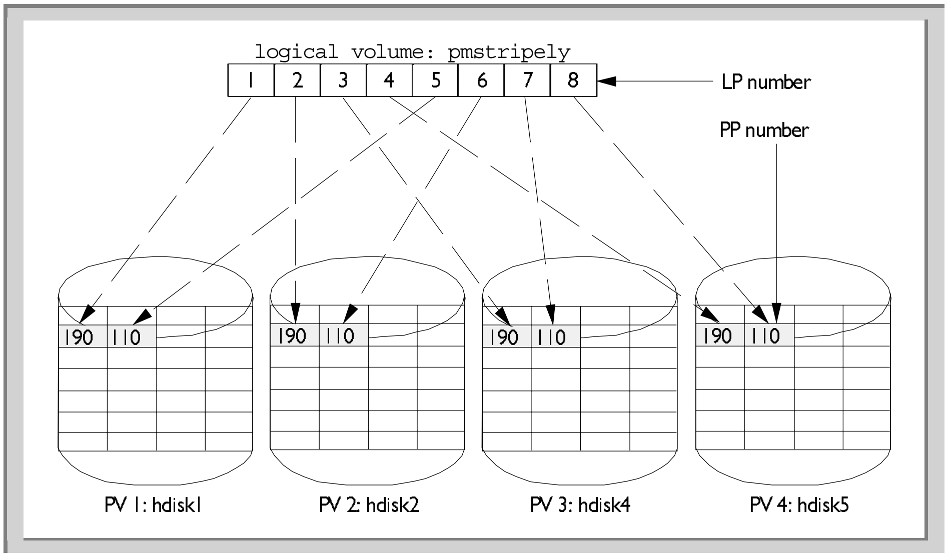


Figure 50. The physical partition allocation of the poor man's stripe

At first look, it looks like a striped logical volume. But, there is a distinct difference between this logical volume and a striped logical volume since there is no chunk level I/O striping benefit. Also, this physical volume allocation does not provide any I/O performance gain, but also it provides vulnerability (any disk fails and the logical volume is lost). This configuration is not beneficial in most circumstances, and it is called a *poor man's stripe* (not an actual striped logical volume).

3.1.2.3 Pre-defined allocation policy

In a striped logical volume, the inter-physical volume allocation policy is forcibly pre-defined as maximum to achieve the logical partitions mapping scheme described in 3.1.1.1, "The logical partitions mapping scheme" on page 138. It cannot be changed after the initial creation. If you try to force this value, then you would see the error message described in "Prohibited options of the mklv command with striping" on page 150.

For real examples of managing striped logical volumes, please refer to 3.2, "Real examples" on page 152.

3.1.3 The reorganization relocation flag

The reorganization relocation flag is also one of the attributes of the logical volumes. It stipulates whether the logical volume can be relocated or not. For non-striped logical volume case, it is set to *y* (relocatable) to allow the relocation of the logical volume during the reorganization process.

The following descriptions are from the command reference of the `mklv` command.

`-r Relocate` Sets the reorganization relocation flag. For striped logical volumes, the `Relocate` parameter must be set to n (the default for striped logical volumes). The Relocate parameter can be one of the following:

`y` Allows the logical volume to be relocated during reorganization. This is the default for relocation.

`n` Prevents the logical volume from being relocated during reorganization.

To confirm the reorganization relocation flag, you can use the `lslv` command:

```
root@lorraine:/ [113] # lslv hd6
LOGICAL VOLUME:     hd6                    VOLUME GROUP:   rootvg
LV IDENTIFIER:      00041631ae592047.2     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       opened/syncd
TYPE:               paging                 WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        16 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                64                     PPs:            64
STALE PPs:          0                      BB POLICY:      non-relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV?: yes
```

In this example, the reorganization relocation flag is set to yes (as shown in the rectangle). Since this is not a striped logical volume, it is useful to allow relocation during the reorganization process. This is the default value for non-striped logical volumes.

3.1.3.1 Relocation flag for striped logical volumes

On the striped logical volumes, the reorganization relocation flag is forcibly pre-defined as no (as shown in the rectangle).

```
root@lorraine:/ [106] # lslv stripelv
LOGICAL VOLUME:     stripelv               VOLUME GROUP:   stripevg
LV IDENTIFIER:      000416313d5cbeb6.1     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            1024                   PP SIZE:        8 megabyte(s)
COPIES:             1                      SCHED POLICY:   striped
LPs:                600                    PPs:            600
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       maximum                RELOCATABLE:    no
INTRA-POLICY:       middle                 UPPER BOUND:    3
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
STRIPE WIDTH:       3
STRIPE SIZE:        64K
```

If you try to force its value to yes, you will see following error message.

```
root@lorraine:/ [151] # chlv -r y stripelv
0516-1042 chlv: Cannot change a striped logical volume to relocatable.
0516-704 chlv: Unable to change logical volume stripelv.
```

This means that the striped logical volumes are never reorganized during the execution of the `reorgvg` command. If the striped logical volume was reorganized, then the physical partitions allocation policy might not fit the logical partitions mapping scheme described in 3.1.1.1, “The logical partitions mapping scheme” on page 138. It would cause the loss of the performance benefit of the striped logical volumes.

3.1.3.2 Reorganization and migration

Reorganization and migration of physical partitions are two different tasks and shouldn’t be confused. The reorganization (executed by the `reorgvg` command) is a procedure that repacks the actual location of physical partitions in one physical volume to generate contiguous allocation of the physical partitions. The migration of physical partitions (executed by the `migratepv` command) is a procedure that moves the entire physical partition from one physical volume to another within a volume group.

This reorganization relocation flag does not prohibit the migration of the physical disk that accommodates the striped logical volume.

3.1.4 Creation and extension of the striped logical volumes

This section provides some information and guidelines about the creation and extension of the striped logical volumes.

- Specify a multiple of the strip width as logical partition number during the initial creation of the striped logical volume.

- You cannot change the stripe width after the initial creation.
- Extension is made by multiples of the strip width.

3.1.4.1 Specify a multiple of the strip width during initial creation

If you attempt to create a striped logical volume, you can only specify a number of the logical partition that is a multiple of the strip width.

For example, in Figure 51, if you attempt to create a striped logical volume on three physical volumes, you can only specify multiplies of three as the logical partition number. If you specified anything else than these values, then it will round up to the next multiple of the width. If you specified two as the logical partition number, then it is round up to three.

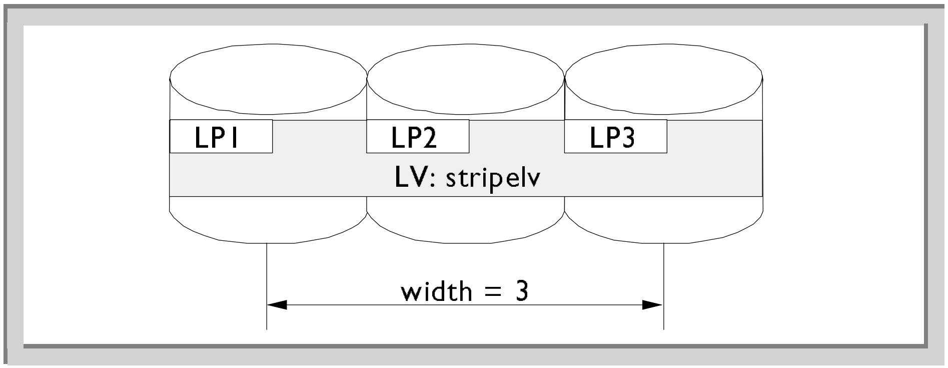


Figure 51. The LP allocation of the striped logical volume

These logical partitions allocated must be equally located on every physical volume composing the striped logical volume. In our example, we have a volume group composed of two physical volumes (see Figure 52). One is 4.5 GB (named hdisk2) and the other 9.1 GB (named hdisk3). We choose 4 MB as the physical partition size for this volume group, then there are 537 PPs on disk 2 and 1084 PPs on disk 3.

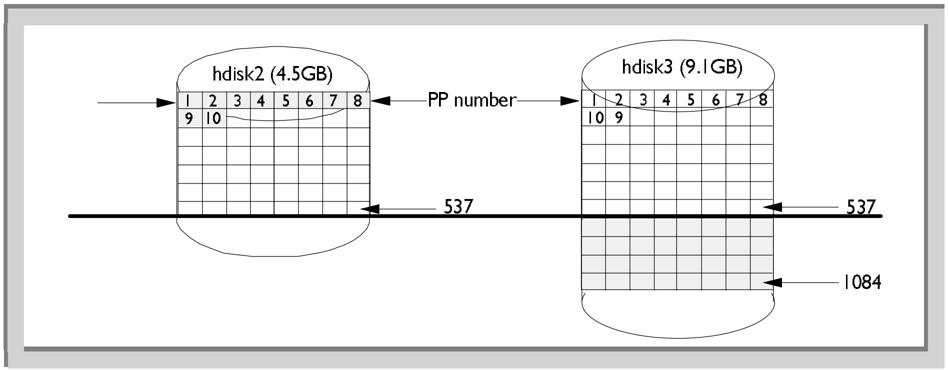


Figure 52. The impossible LP allocation in a striped logical volume

In this example, if we create a striped logical volume in this volume group, then only 1074 logical partitions (537 physical partitions per each physical volume) can be allocated to this striped logical volume. Otherwise, the attempt to create the striped logical volumes will fail (see “Erroneous situations” on page 161). The remaining space of the physical volume hdisk3 (shown as the hatched portion under the horizontal bold line in Figure 52) cannot be used in striped logical volume. You can still use this space for non-striped logical volumes, but this can affect the I/O performance of the striped logical volumes.

3.1.4.2 Cannot change the stripe width

You cannot add new physical volumes to an already created striped volume (see Figure 53). To do this, you have to recreate it (you have to create a backup of it and restore it). Because the actual data is already striped on the physical volumes, if new physical volumes are added to this striped logical volume, all the data that reside on physical volumes have to be re-striped to the newly added physical volume physical partitions.

Note

You cannot add new physical volume to already-created striped logical volumes. In another words, you cannot change the stripe width of an existing striped logical volume.

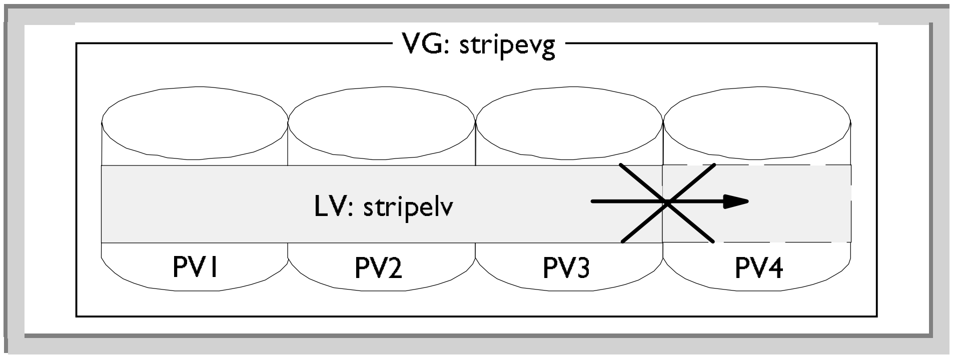


Figure 53. Impossible extension of the striped logical volume

3.1.4.3 Extension is made by stripe width base

To extend the striped logical volumes, you have to specify a multiple of the *stripe width* as logical partition number (see “How to extend the striped logical volumes” on page 158). Otherwise the attempt to extend the striped logical volumes will fail (see Figure 54). In this case, since the striped logical volume scatters over three physical volumes, the stripe width is three. Therefore, you have to specify a multiple of three as the extension logical partition number.

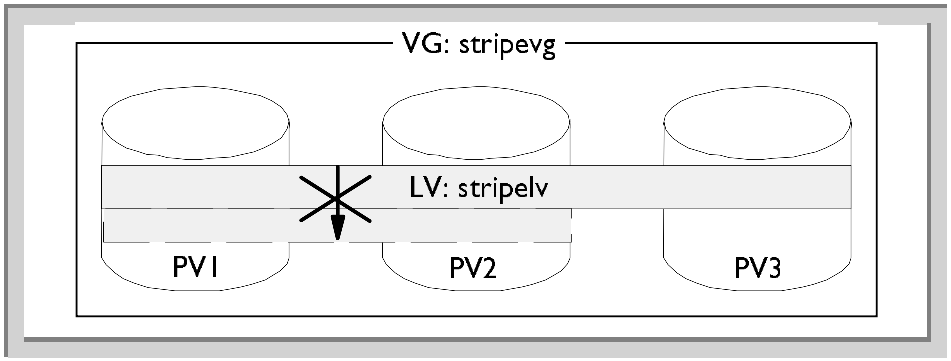


Figure 54. Impossible extension of the striped logical volume with wrong width

3.1.5 Prohibited options of the mklv command with striping

To create the striped logical volume, you must use the -S option of the `mklv` command. This optional flag specifies that the logical volume is designated as a striped logical volume. There is a fundamental difference between the mirroring and striping function, of course, other than the functionality itself. You can always mirror an existing non-mirrored logical volume (including a striped logical volume from AIX Version 4.3.3), and also remove the mirror from the mirrored logical volumes. But, you cannot convert a non-striped logical volume to a striped logical volume, or vice versa. The only way to create the striped logical volumes is to explicitly specify the -S flag on the `mklv` command line or use the corresponding SMIT panel.

The AIX LVM provides many ways to control the physical partitions allocation of the logical volumes. They are forced by optional flags of the `mklv` command. But, if you attempt to create a striped logical volume, some of these optional flags cannot be used with the `-S` flag. For example:

- On AIX Version 4.3.3:

```
root@lorraine:/ [168] # mklv -y stripelv -e x -S 64K workvg 2 hdisk2 hdisk4
0516-1240 mklv: The -e, -d, -m, -s options cannot be
        used with striped logical volumes.
Usage: mklv [-a IntraPolicy] [-b BadBlocks] [-c Copies] [-d Schedule]
        [-e InterPolicy] [-i] [-L Label] [-m MapFile] [-r Relocate] [-s Strict]
        [-t Type] [-u UpperBound] [-v Verify] [-w MWC] [-x MaxLPs] [-y LVname]
        [-Y Prefix] [-S StripeSize] [-U userid] [-G groupid] [-P modes]
        VGname NumberOfLPs [PVname...]
Makes a logical volume.
Note:   You must be a root user to set the IDs and modes
        and these are applicable for big volume group format LVs only.
```

- On AIX Version 4.3.2:

```
root@pukupuku:/ [251] # mklv -y stripelv -e x -S 64K workvg 2 hdisk2 hdisk4
0516-1032 mklv: The -e, -m, -s, -u, -w, -d and -c options cannot be
        used with striped logical volumes (-S option).
Usage: mklv [-a IntraPolicy] [-b BadBlocks] [-c Copies] [-d Schedule]
        [-e InterPolicy] [-i] [-L Label] [-m MapFile] [-r Relocate] [-s Strict]
        [-t Type] [-u UpperBound] [-v Verify] [-w MWC] [-x MaxLPs] [-y LVname]
        [-Y Prefix] [-S StripeSize] [-U userid] [-G groupid] [-P modes]
        VGname NumberOfLPs [PVname...]
Makes a logical volume.
Note:   You must be a root user to set the IDs and modes
        and these are applicable for big volume group format LVs only.
```

The exclusive optional flags of the -S flag for the `mklv` command are:

Table 13. The prohibited optional flags with -S of the mklv command

AIX Version	Prohibited optional flags with -S
4.1 to 4.3.2	-c, -d, -e, -m, -s, -u, -w
4.3.3	-e, -d, -m, -s

The differences shown for Version 4.3.3 come from the support of the mirroring and striping function (see 3.3, “The mirror and stripe function” on page 162).

As a result of these exclusive optional flags, you do not have the ability to precisely control the physical partitions allocation of the logical volumes.

3.1.6 Prohibited options of the extendlv command with striping

Some limitations also apply to the extendlv command, used to extend the logical volume size. If you attempt to extend the striped logical volumes, the following two command options are prohibited.

- -m mapfile option
- Physical volume names

3.1.7 Summary

As a conclusion for this section, it is recommended that you keep in mind the following rules.

- Due to the lack of precise control for the physical partitions allocation, you may not place the striped logical volumes as you want. The best way to avoid this situation is to dedicate a volume group that accommodates the striped logical volumes. It also benefits the I/O performance.
- If you create a striped logical volume, then you should choose disks that have same characteristics (especially size). Otherwise, you cannot use the entire surface of the physical volumes for the striped logical volumes.
- If you cannot avoid using different sized physical volumes for creating the striped logical volumes, and, if you have to use the rest of the space of the larger size physical volume(s), you should minimize the I/O activity on that portion. Otherwise, that activity might affect the striped logical I/O performance and you might loose the benefit of the striped logical volumes.

3.2 Real examples

This section focuses on real examples about managing striped logical volumes.

3.2.1 Example configurations

All the examples shown in this section are examined in the same environment as the one seen in“Example of configurations” on page 142.

```
root@lorraine:/ [259] # lsdev -Cc disk
hdisk0 Available 10-60-00-8,0  16 Bit SCSI Disk Drive
hdisk1 Available 10-60-00-9,0  16 Bit LVD SCSI Disk Drive
hdisk2 Available 10-60-00-10,0 16 Bit LVD SCSI Disk Drive
hdisk3 Available 30-58-00-8,0  16 Bit SCSI Disk Drive
hdisk4 Available 30-58-00-9,0  16 Bit LVD SCSI Disk Drive
hdisk5 Available 30-58-00-10,0 16 Bit LVD SCSI Disk Drive
root@lorraine:/ [260] # mkvg -y stripevg -s 8 -t 2 hdisk2
0516-1193 mkvg: WARNING, once this operation is completed, volume group stripevg
        cannot be imported into AIX 430 or lower versions. Continue (y/n)?
```

```
y
stripevg
root@lorraine:/ [261] # extendvg stripevg hdisk4
root@lorraine:/ [262] # extendvg stripevg hdisk5
root@lorraine:/ [272] # lsvg
stripevg
rootvg
root@lorraine:/ [273] # lspv
hdisk0         0004163128f3de5a    rootvg
hdisk1         000416314bd724bc    None
hdisk2         0004163192b1d7f5    stripevg
hdisk3         000416314bd749f8    None
hdisk4         000416314bdada38    stripevg
hdisk5         0004163192b1d297    stripevg
root@lorraine:/ [274] # lsvg stripevg
VOLUME GROUP:   stripevg                 VG IDENTIFIER:  000416313d581469
VG STATE:       active                   PP SIZE:        8 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      1611 (12888 megabytes)
MAX LVs:        256                      FREE PPs:       1611 (12888 megabytes)
LVs:            0                        USED PPs:       0 (0 megabytes)
OPEN LVs:       0                        QUORUM:         2
TOTAL PVs:      3                        VG DESCRIPTORS: 3
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     3                        AUTO ON:        yes
MAX PPs per PV: 1016                     MAX PVs:        32
```

Note

In this example, the stripevg volume group is created with the -t factor as 2. You can see this from the `mkvg` command line options.

3.2.2 How to create the striped logical volumes

To create a striped logical volume, you can use the SMIT interface or the `mklv` command directly. In either case, there are three ways to specify the necessary information to create a striped logical volume.

- Specify all physical volume names directly.
- Specify the maximum number of physical volumes allocated to the striped logical volume.
- Specify both of the options above.

The -u option of the `mklv` command is used to specify the maximum number of physical volumes used for the physical partition allocation of this striped logical volume. It is described in the command reference of the `mklv` command as follow:

-u *UpperBound* Set the maximum number of physical volumes for new allocation. The value of the *Upperbound* variable should be between one and the total number of physical volumes. When using striped logical volumes or super strictness, the upper bound indicates the maximum number of physical volumes allowed for each mirror copy.

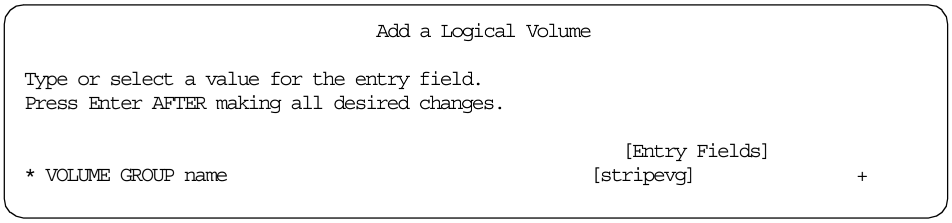
> **Note**
>
> When creating a super strict logical volume, you must specify physical volumes or use the -u flag.

You might get confused on which options are taken if both options are specified during the initial creation of a striped logical volume. However, the rule is simple. The -u option is just ignored if both options are specified.

3.2.2.1 Specifying the physical volume names

In this case, you have to explicitly specify the physical volume names for the striped logical volume. The following example shows how to create a logical volume using SMIT.

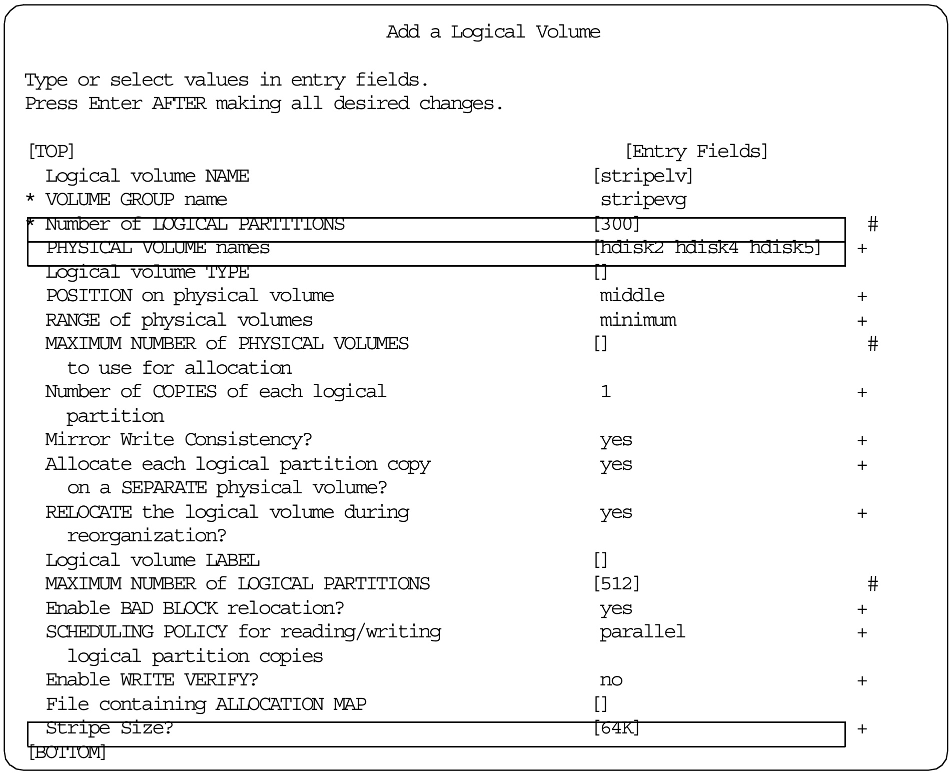
```
root@lorraine:/ [288] # smit lv
  Add a Logical Volume
```



```
                              Add a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                      [Entry Fields]
* VOLUME GROUP name                                 [stripevg]          +
```

In the following dialog, you have to specify the following values in addition to the logical volume name.

- Number of LOGICAL PARTITIONS
- PHYSICAL VOLUME names
- Stripe size



```
                                 Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
  Logical volume NAME                                 [stripelv]
* VOLUME GROUP name                                    stripevg
* Number of LOGICAL PARTITIONS                        [300]                #
  PHYSICAL VOLUME names                               [hdisk2 hdisk4 hdisk5]  +
  Logical volume TYPE                                 []
  POSITION on physical volume                          middle              +
  RANGE of physical volumes                            minimum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  []                   #
    to use for allocation
  Number of COPIES of each logical                     1                   +
    partition
  Mirror Write Consistency?                            yes                 +
  Allocate each logical partition copy                 yes                 +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                   yes                 +
    reorganization?
  Logical volume LABEL                                []
  MAXIMUM NUMBER of LOGICAL PARTITIONS                [512]                #
  Enable BAD BLOCK relocation?                         yes                 +
  SCHEDULING POLICY for reading/writing                parallel            +
    logical partition copies
  Enable WRITE VERIFY?                                 no                  +
  File containing ALLOCATION MAP                      []
  Stripe Size?                                        [64K]                +
[BOTTOM]
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [290] # mklv -y stripelv -S 64K stripevg 300 hdisk2 hdisk4 hdisk5
```

After the execution of this command, you will have the following physical partition allocation map for the striped logical volume. This result is also represented in Figure 55 on page 156.

```
root@lorraine:/ [292] # lslv -m stripelv
stripelv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0109 hdisk2
0002  0109 hdisk4
0003  0109 hdisk5
0004  0110 hdisk2
0005  0110 hdisk4
0006  0110 hdisk5
(snipped many lines)
0295  0207 hdisk2
0296  0207 hdisk4
0297  0207 hdisk5
0298  0208 hdisk2
0299  0208 hdisk4
0300  0208 hdisk5
```

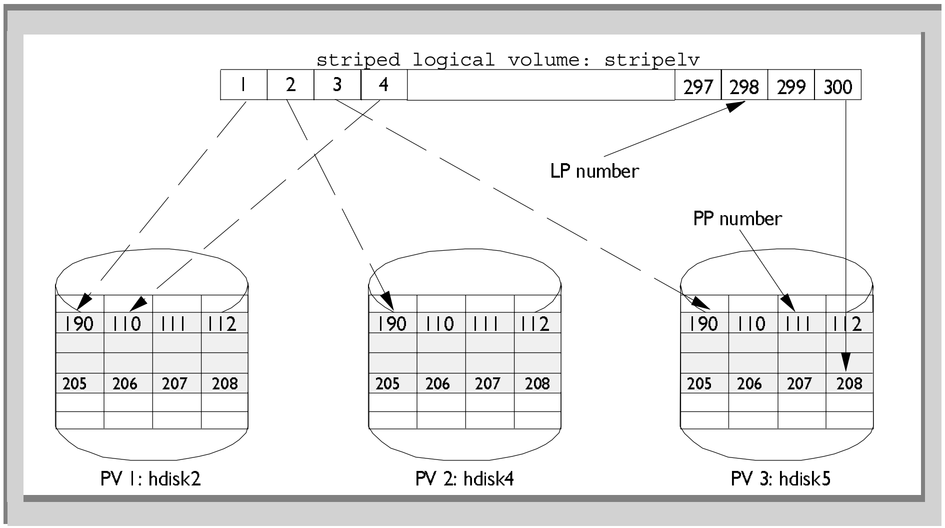


Figure 55. The LVs and the PVs mapping on the striped LV

3.2.2.2 Specifying the maximum number of physical volumes

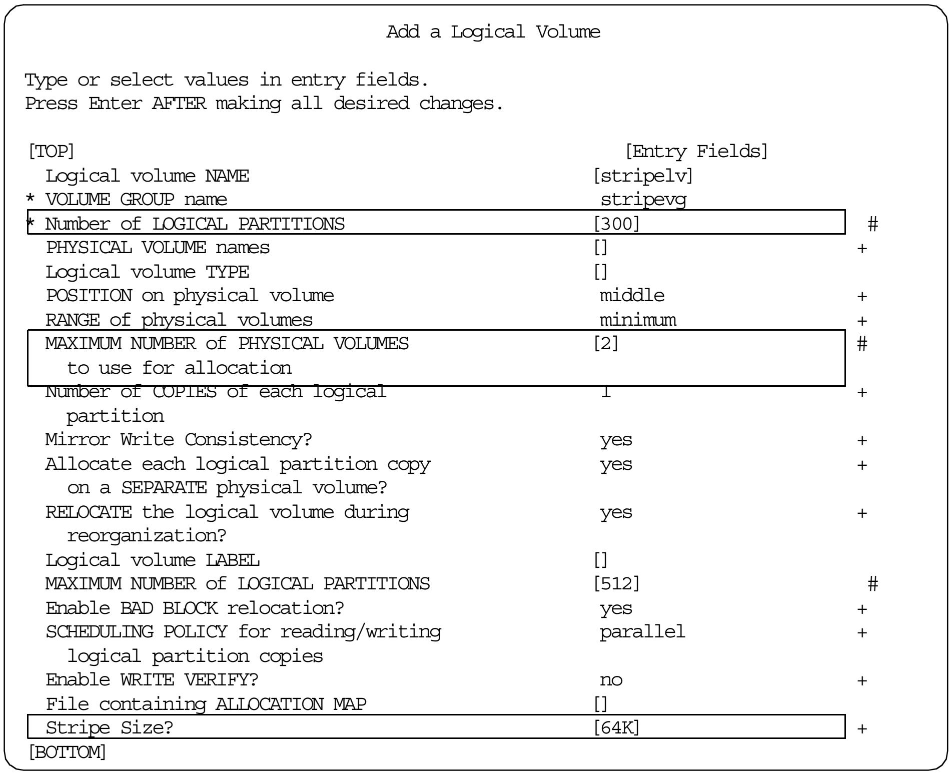
In this case, you can specify the number of physical volumes that compose the logical volume. If you specify a number smaller than the number of the physical volumes in the volume group, the AIX LVM sequentially selects physical volumes up to the number specified. The following example shows how to create the logical volume using SMIT.

```
root@lorraine:/ [288] # smit lv
  Add a Logical Volume
```

```
                                    Add a Logical Volume

 Type or select a value for the entry field.
 Press Enter AFTER making all desired changes.

                                                             [Entry Fields]
 * VOLUME GROUP name                                       [stripevg]              +
```

In the following dialog, you have to specify the following values in addition to the logical volume name.

- Number of LOGICAL PARTITIONS
- MAXIMUM NUMBER of PHYSICAL VOLUMES to use for allocation
- Stripe size



```
                              Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                  [Entry Fields]
  Logical volume NAME                                 [stripelv]
* VOLUME GROUP name                                    stripevg
* Number of LOGICAL PARTITIONS                        [300]              #
  PHYSICAL VOLUME names                               []                +
  Logical volume TYPE                                 []
  POSITION on physical volume                          middle           +
  RANGE of physical volumes                            minimum          +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  [2]               #
    to use for allocation
  Number of COPIES of each logical                     1                +
    partition
  Mirror Write Consistency?                            yes              +
  Allocate each logical partition copy                 yes              +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                   yes              +
    reorganization?
  Logical volume LABEL                                []
  MAXIMUM NUMBER of LOGICAL PARTITIONS                [512]              #
  Enable BAD BLOCK relocation?                         yes              +
  SCHEDULING POLICY for reading/writing                parallel         +
    logical partition copies
  Enable WRITE VERIFY?                                 no               +
  File containing ALLOCATION MAP                      []
  Stripe Size?                                        [64K]             +
[BOTTOM]
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [290] # mklv -y stripelv -u 2 -S 64K stripevg 300
```

Here is the result of the execution of this command:

```
root@lorraine:/ [294] # lslv -m stripelv
stripelv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0109 hdisk2
0002  0109 hdisk4
0003  0110 hdisk2
0004  0110 hdisk4
(snipped many lines)
0297  0257 hdisk2
0298  0257 hdisk4
0299  0258 hdisk2
0300  0258 hdisk4
```

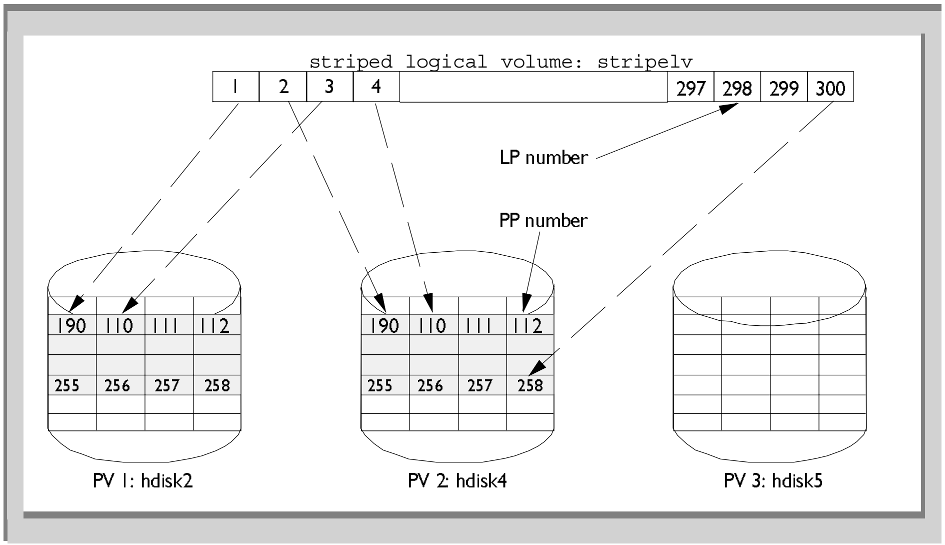


Figure 56. The LVs and the PVs mapping on the striped LV (2 PV case)

If you specify both the -u option and the physical volume names at the `mklv` command execution, then the -u option is ignored. For example, if you invoke:

```
root@lorraine:/ [290] # mklv -y stripelv -u 2-S 64K stripevg 300 hdisk2 hdisk4 hdisk5
```

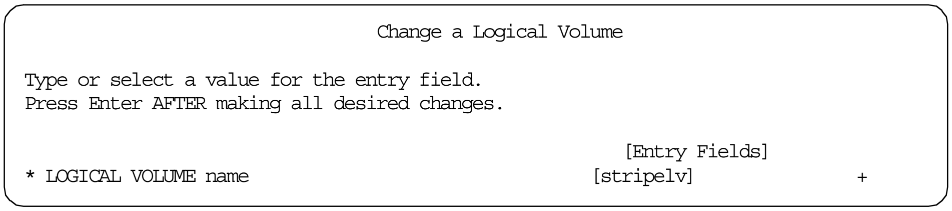
The result of the above execution will show the same physical partition allocation map than the striped logical volume shown in 3.2.2.1, “Specifying the physical volume names” on page 154.

3.2.3 How to extend the striped logical volumes

To extend the logical volumes (not only the striped one), you may have to increase the maximum logical partition number first (if the new logical volume size is greater than 512 logical partitions).

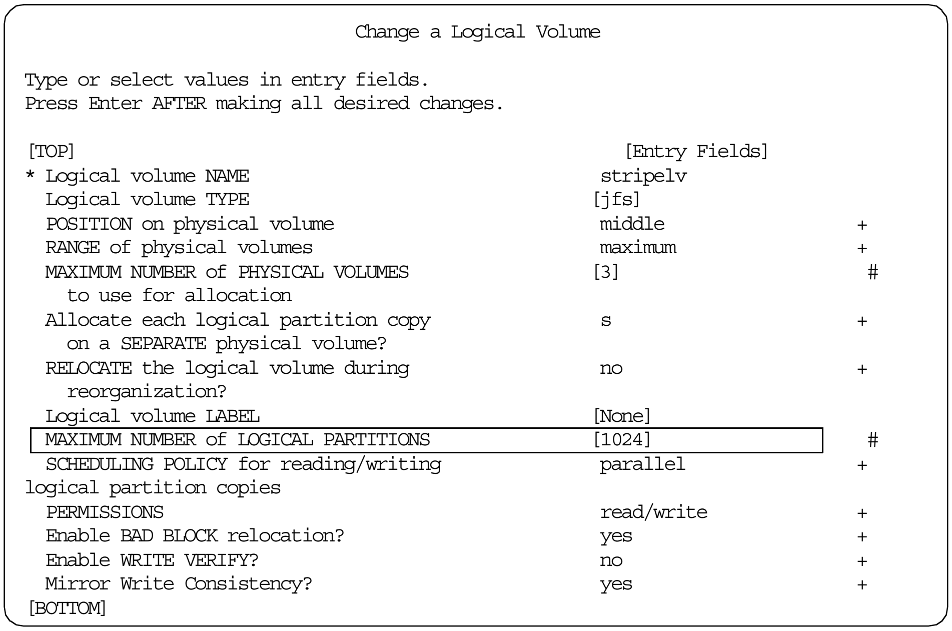
In the following example, the maximum number of logical partition is increased to 1024 using the SMIT interface.

```
root@lorraine:/ [288] # smit lv
  Set Characteristic of a Logical Volume
    Change a Logical Volume
```



```
                                 Change a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                     [Entry Fields]
* LOGICAL VOLUME name                             [stripelv]            +
```

Then enter the appropriate number as MAXIMUM NUMBER of LOGICAL PARTITIONS. This number should be equal or greater than the logical partition number of the striped logical volumes after its extension. In this example, we specified 1024.



```
                                Change a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                [Entry Fields]
* Logical volume NAME                              stripelv
  Logical volume TYPE                             [jfs]
  POSITION on physical volume                      middle               +
  RANGE of physical volumes                        maximum              +
  MAXIMUM NUMBER of PHYSICAL VOLUMES              [3]                    #
    to use for allocation
  Allocate each logical partition copy             s                    +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during               no                   +
    reorganization?
  Logical volume LABEL                            [None]
  MAXIMUM NUMBER of LOGICAL PARTITIONS            [1024]                 #
  SCHEDULING POLICY for reading/writing            parallel             +
logical partition copies
  PERMISSIONS                                      read/write           +
  Enable BAD BLOCK relocation?                     yes                  +
  Enable WRITE VERIFY?                             no                   +
  Mirror Write Consistency?                        yes                  +
[BOTTOM]
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [331] # chlv -x 1024 stripelv
```

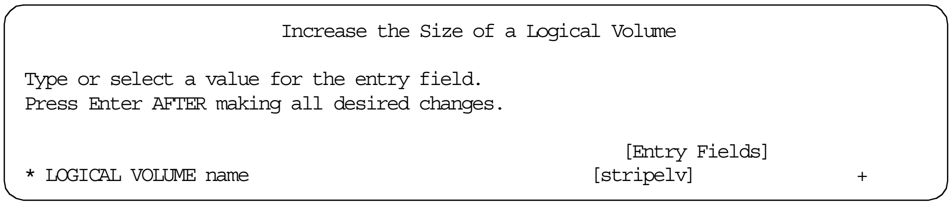
You can examine the result of this execution (shown as MAX LPs):

```
root@lorraine:/ [332] # lslv stripelv
LOGICAL VOLUME:     stripelv                VOLUME GROUP:   stripevg
LV IDENTIFIER:      000416313d5cbeb6.1      PERMISSION:     read/write
VG STATE:           active/complete         LV STATE:       closed/syncd
TYPE:               jfs                     WRITE VERIFY:   off
MAX LPs:            1024                    PP SIZE:        8 megabyte(s)
COPIES:             1                       SCHED POLICY:   striped
LPs:                300                     PPs:            300
```

```
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       maximum                RELOCATABLE:    no
INTRA-POLICY:       middle                 UPPER BOUND:    3
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
STRIPE WIDTH:       3
STRIPE SIZE:        64K
```

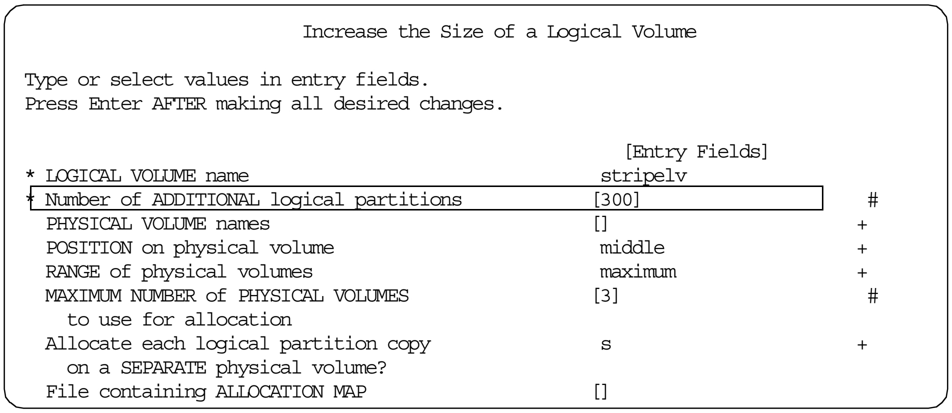
The following example presents how to extend the striped logical volume by 300 logical partitions (it is a hundred times the stripe width, 3).

```
root@lorraine:/ [288] # smit lv
  Set Characteristic of a Logical Volume
    Increase the Size of a Logical Volume
```



```
                          Increase the Size of a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* LOGICAL VOLUME name                                 [stripelv]           +
```

Then input the appropriate number as Number of ADDITIONAL logical partitions. In this example, we input 300.



```
                            Increase the Size of a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* LOGICAL VOLUME name                                  stripelv
* Number of ADDITIONAL logical partitions             [300]                 #
  PHYSICAL VOLUME names                               []                   +
  POSITION on physical volume                          middle              +
  RANGE of physical volumes                            maximum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  [3]                   #
    to use for allocation
  Allocate each logical partition copy                 s                   +
    on a SEPARATE physical volume?
  File containing ALLOCATION MAP                      []
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [106] # extendlv stripelv 300
```

You can examine the result of this execution like this (shown as LPs):

```
root@lorraine:/ [106] # lslv stripelv
LOGICAL VOLUME:     stripelv               VOLUME GROUP:   stripevg
LV IDENTIFIER:      000416313d5cbeb6.1     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            1024                   PP SIZE:        8 megabyte(s)
COPIES:             1                      SCHED POLICY:   striped
LPs:                600                    PPs:            600
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       maximum                RELOCATABLE:    no
INTRA-POLICY:       middle                 UPPER BOUND:    3
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
STRIPE WIDTH:       3
STRIPE SIZE:        64K
```

3.2.4 Erroneous situations

To extend the striped logical volumes, you have to specify a multiple of the stripe width as the logical partition number. Otherwise, you will see following the error message.

```
root@lorraine:/ [101] # extendlv stripelv 1
0516-1036 extendlv: Striped logical volume size can only be an even
        multiple of the striping width.
0516-788 extendlv: Unable to extend logical volume.
Usage: extendlv [-a IntraPolicy] [-e InterPolicy] [-m MapFile]
        [-s Strict] [-u UpperBound] LVname NumberOfLPs [PVname...]
Extends the size of a logical volume.
```

You cannot specify the physical volume names as command line options of the `extendlv` command either. Otherwise, you will see following the error message.

```
root@lorraine:/ [103] # extendlv stripelv 4 hdisk2
0516-1040 extendlv: Physical volumes cannot be specified when extending the
striped logical volume.
Usage: extendlv [-a IntraPolicy] [-e InterPolicy] [-m MapFile]
        [-s Strict] [-u UpperBound] LVname NumberOfLPs [PVname...]
Extends the size of a logical volume.
```

To overcome this situation, you have to remove the physical volume name on the command line.

```
root@lorraine:/ [104] # extendlv stripelv 4
```

If you attempt to create and/or extend striped logical volumes beyond the point where physical partitions cannot be allocated anymore (shown as bold line in Figure 57 on page 162), then you will see the following error message.

```
root@lorraine:/ [123] # mklv -y stripe2lv -S 64K stripevg 1076  hdisk2 hdisk3
0516-1034 mklv: Not enough physical partitions in physical volume hdisk2.
0516-822 mklv: Unable to create logical volume.
```

To overcome this situation, you should not specify more than 1074 logical partition size on the command line.

```
root@lorraine:/ [124] # mklv -y stripe2lv -S 64K stripevg 1074 hdisk2 hdisk3
```

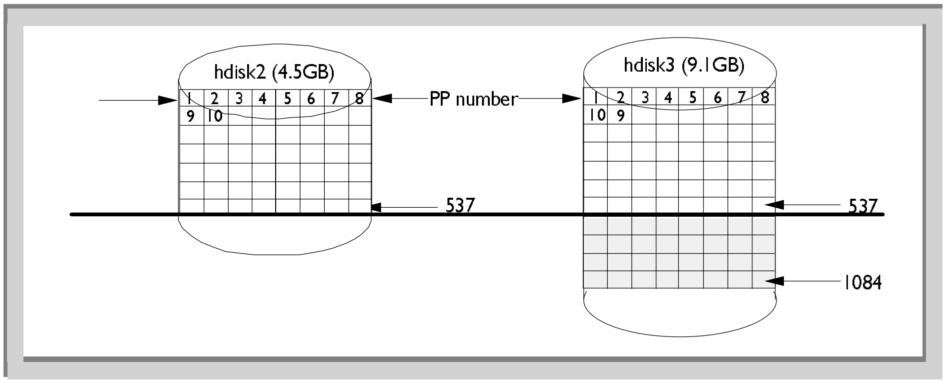


Figure 57. Erroneous situation

3.3 The mirror and stripe function

AIX Version 4.3.3 introduces the new mirroring and striping function (also known as RAID 0+1 or RAID 10 technology). This function provides better data availability at the cost of extra disks.

In the case of non-mirrored and striped logical volume, failure of one physical volume causes the loss of the whole striped logical volume. There is no redundancy. The mirroring and striping function prevents this situation from happening by adding up to three mirrors to the striped logical volumes. One failing disk in a striped mirror copy does not bring down the entire logical volume. The remaining (working) disks in the striped mirror copy continue to service striped units. Although the striped units on the failed disk are inaccessible, the mirrored copies of these units are available from another mirror. In this case, users are unaware that a disk is unavailable.

To support the mirroring and striping function, a new partition allocation policy (*super strict*) is introduced in AIX Version 4.3.3. This policy prohibits partitions from one mirror from sharing a disk with a second or third mirror. But, why is the super strict allocation policy needed for the mirror and stripe function?

Note

The super strict policies and the mirroring and striping function are not supported on AIX releases prior to AIX Version 4.3.3. You cannot import volume groups to AIX Version 4.3.2 and lower if the volume group includes this function.

3.3.1 Super strict allocation policy

AIX Version 4.3.3 introduces this new allocation policy type, super strict. This option is specified to the `mklv` command as the s flag for the -s option. Here are the possible flags for the strict option:

-s Strict Determines the strict allocation policy. Copies of a logical volume can be allocated to share or not to share the same physical volume. The strict parameter is represented by one of the following:

- y Sets a strict allocation, so copies for a logical partition cannot share the same physical volume. This is the default for allocation policy.
- n Does not set a strict allocation policy. So copies for a logical partition cannot share the same physical volume.
- s Sets a super strict allocation policy, so that the partitions allocated for one mirror copy cannot share a physical volume with the partitions from another mirror copy.

3.3.1.1 Create the poor man's stripe first, then mirror

Let us go back to the poor man's stripe (3.1.2.2, "The poor man's stripe" on page 143). If you create the poor man's striped logical volume (which scatters over two physical volumes), then create a mirror copy, you will have the following allocation for the physical partitions (see Figure 58 on page 164).

```
root@lorraine:/ [159] # mklv -y pmstripelv -e x pmstripevg 300 hdisk1 hdisk2
pmstripelv
root@lorraine:/ [163] # mklvcopy pmstripelv 2 hdisk4 hdisk5
root@lorraine:/ [164] # lslv -m pmstripelv
pmstripelv:N/A
LP    PP1  PV1              PP2  PV2              PP3  PV3
0001  0109 hdisk1           0109 hdisk4
0002  0109 hdisk2           0109 hdisk5
0003  0110 hdisk1           0110 hdisk4
0004  0110 hdisk2           0110 hdisk5
(many lines are snipped off)
0297  0257 hdisk1           0257 hdisk4
0298  0257 hdisk2           0257 hdisk5
0299  0258 hdisk1           0258 hdisk4
```

From a system management perspective, this configuration is not bad (actually, this allocation is the same as the one shown in Figure 62 on page 174). If either PV1 and PV2, or PV3 and PV4 fail, there is still a valid copy.

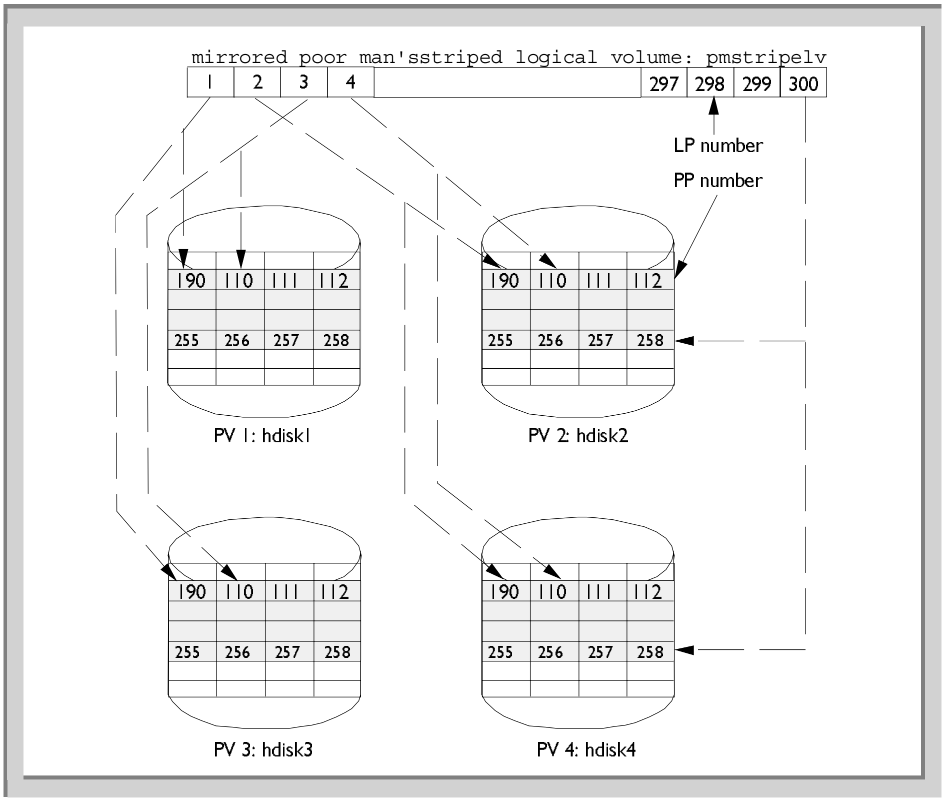


Figure 58. Create the poor man's stripe with width 2, then mirror it

3.3.1.2 Create poor man's stripe with mirror upon initial creation

However, if you create the poor man's striped logical volume (which scatters over two physical volumes) with mirror during the initial creation, then you will have the following allocation of the physical partitions (see Figure 59 on page 165).

```
root@lorraine:/ [170] # mklv -y mpmstripelv -e x -c 2 pmstripevg 4 hdisk1 hdisk2 hdisk4
hdisk5
root@lorraine:/ [172] # lslv -m mpmstripelv
mpmstripelv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0109 hdisk1            0110 hdisk2
0002  0109 hdisk2            0110 hdisk4
0003  0109 hdisk4            0110 hdisk5
0004  0109 hdisk5            0110 hdisk1
```

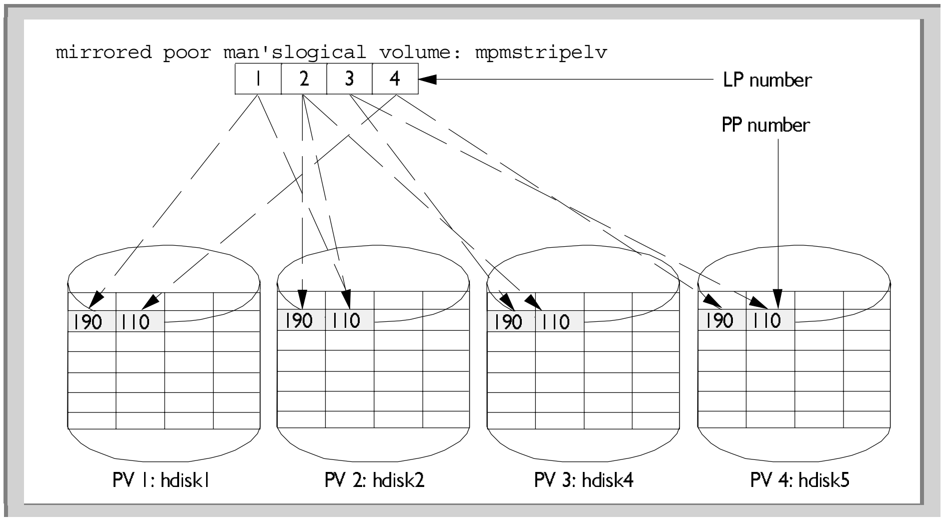


Figure 59. Create poor man's stripe with width 2 with mirror upon initial creation

This is a very bad configuration. Any two disks failing will cause the loss of the entire logical volume. It is difficult to add or remove a mirror to or from this allocation, since each copy shares its physical volumes with another mirror (each mirror is represented with rectangles of different patterns in Figure 59).

The reason for this allocation is very simple; you instructed the system to do so. Since you specified both the -e x option (inter-physical volume allocation set to maximum) and the -c 2 option (create two copy upon initial creation), then AIX LVM attempts to do its best. This bad configuration is not desirable, but it still conforms to the strict allocation policy (shown in the rectangle in the following example), since every copy of the specific logical partition exists on a separate physical volume.

```
root@lorraine:/ [173] # lslv mpmstripelv
LOGICAL VOLUME:     mpmstripelv            VOLUME GROUP:   pmstripevg
LV IDENTIFIER:      00041631540cc083.1     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        8 megabyte(s)
COPIES:             2                      SCHED POLICY:   parallel
LPs:                4                      PPs:            8
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       maximum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV?: yes
```

In another words, the strict allocation policy cannot prohibit this bad configuration. This is the reason why supporting the mirror and stripe function requires a super strict allocation policy.

3.3.1.3 Initial creation with mirror and super strict allocation policy

If you specified the super strict allocation policy in the previous example, you will have the following allocation of the physical partitions (see Figure 60). You can force the super strict allocation policy with the -S s option on the command line.

```
root@lorraine:/ [180] # mklv -y ssmpmstripelv -e x -c 2 -s s pmstripevg 4 hdisk1 hdisk2
hdisk4 hdisk5
root@lorraine:/ [180] # lslv -m ssmpmstripelv
smpmstripelv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0109 hdisk1            0109 hdisk2
0002  0109 hdisk4            0109 hdisk5
0003  0110 hdisk1            0110 hdisk2
0004  0110 hdisk4            0110 hdisk5
```

To confirm the super strict allocation policy, you can use the `lslv` command:

```
root@lorraine:/ [281] # lslv ssmpmstripelv
LOGICAL VOLUME:     ssmpmstripelv          VOLUME GROUP:   pmstripevg
LV IDENTIFIER:      00041631540cc083.1     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        8 megabyte(s)
COPIES:             2                      SCHED POLICY:   parallel
LPs:                4                      PPs:            8
STALE PPs:          0                      BB POLICY:      relocatable
INTER-POLICY:       maximum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    2
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: on
EACH LP COPY ON A SEPARATE PV ?: yes
```

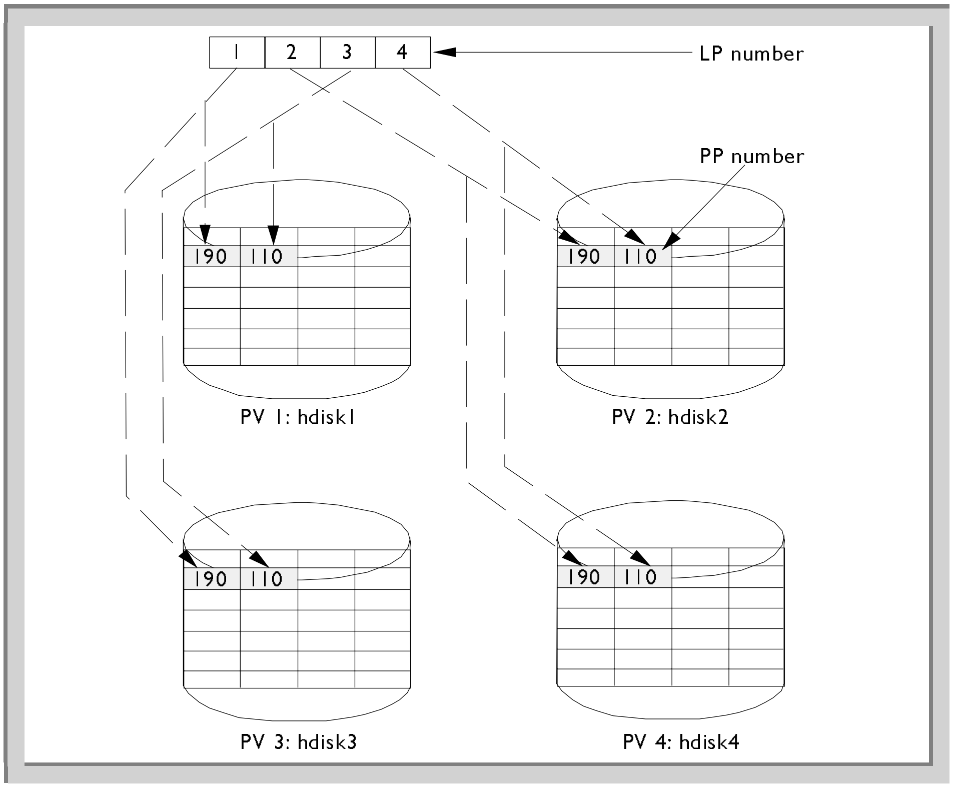


Figure 60. Mirror with super strict allocation policy

This allocation is almost the same as the one shown in “Create the poor man’s stripe first, then mirror” on page 163, except for the allocated physical partition number. This allocation is not only desirable, but also necessary for a correct management of a mirror and stripe logical volume.

The only way to guarantee the allocation of the physical partition represented in Figure 60 is by adopting the super strict allocation policy for the mirrored and striped logical volumes. In fact, this is the default value for the newly created striped logical volume, and it forces the super strict style allocation. Using this facility, each mirror (shown with a different pattern of rectangles in Figure 61) does not share the physical volumes with other mirrors. It enables you to add or remove mirror(s) from the striped logical volumes.

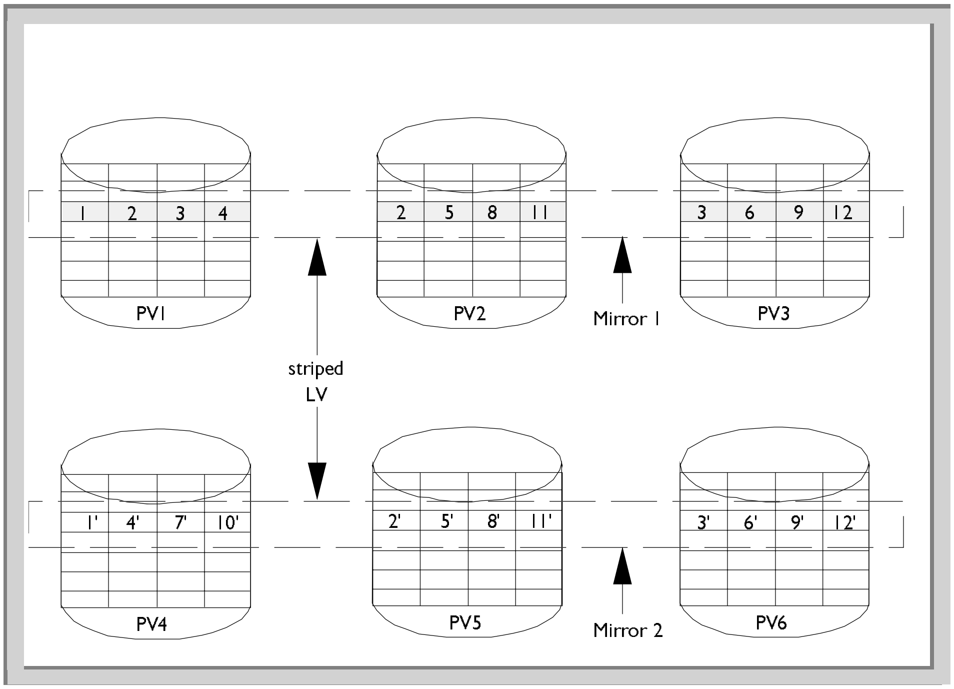


Figure 61. Striped logical volume (mirrored with super strict)

3.3.2 How to manage a mirrored and striped logical volume

This section shows real examples of managing mirrored and striped logical volumes.

3.3.2.1 Example configurations

All the examples shown in this section are run with the same environment as the one used in “Example of configurations” on page 142. The following configurations are also made.

```
root@lorraine:/ [390] # lsdev -Cc disk
hdisk0 Available 10-60-00-8,0  16 Bit SCSI Disk Drive
hdisk1 Available 10-60-00-9,0  16 Bit LVD SCSI Disk Drive
hdisk2 Available 10-60-00-10,0 16 Bit LVD SCSI Disk Drive
hdisk3 Available 30-58-00-8,0  16 Bit SCSI Disk Drive
hdisk4 Available 30-58-00-9,0  16 Bit LVD SCSI Disk Drive
hdisk5 Available 30-58-00-10,0 16 Bit LVD SCSI Disk Drive
root@lorraine:/ [391] # mkvg -y raid10vg -s 8 -t 2 hdisk1
0516-1193 mkvg: WARNING, once this operation is completed, volume group raid10vg
        cannot be imported into AIX 430 or lower versions. Continue (y/n)?
y
raid10vg
root@lorraine:/ [392] # extendvg raid10vg hdisk2
root@lorraine:/ [393] # extendvg raid10vg hdisk4
root@lorraine:/ [394] # extendvg raid10vg hdisk5
```

```
root@lorraine:/ [395] # lsvg
stripevg
rootvg
root@lorraine:/ [396] # lspv
hdisk0         0004163128f3de5a    rootvg
hdisk1         000416314bd724bc    raid10vg
hdisk2         0004163192b1d7f5    raid10vg
hdisk3         000416314bd749f8    temp
hdisk4         000416314bdada38    raid10vg
hdisk5         0004163192b1d297    raid10vg
root@lorraine:/ [397] # lsvg raid10vg
VOLUME GROUP:   raid10vg                 VG IDENTIFIER:  000416313d5cbeb6
VG STATE:       active                   PP SIZE:        8 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      2148 (17184 megabytes)
MAX LVs:        256                      FREE PPs:       2148 (17184 megabytes)
LVs:            0                        USED PPs:       0 (0 megabytes)
OPEN LVs:       0                        QUORUM:         3
TOTAL PVs:      4                        VG DESCRIPTORS: 4
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     4                        AUTO ON:        yes
MAX PPs per PV: 2032                     MAX PVs:        16
```

3.3.2.2 Real example of the creation

To create a mirrored and striped logical volume, you can choose one of the following ways:

- Create the striped logical volume, then mirror and synchronize it.

  Create the striped logical volume (not mirrored yet) first, then create the mirror of this logical volume later. In this case, you have to instruct to synchronize each copy explicitly.

- Specify to create the mirrors during the initial creation.

  Specify to create the mirrors of the striped logical volume upon the initial creation time.

The second way is the easiest to achieve this task. But in the striped logical volume case, due to the lack of ability to precisely control the physical partitions allocation, it is hard to achieve the physical partition allocation that you want to reach, except for a brand new installation.

Thus, the recommended way is the first one. Carefully plan the physical partition allocation of the striped logical volume, then create it, and, finally, mirror it.

Create the striped logical volume, then mirror and synchronize it

1. Create the striped logical volume.

   To create a striped logical volume, you have to explicitly specify the following information.

   a. Where you want to create the striped logical volumes

   You have to choose one of the followings.

   1. The physical volume names.

      You have to specify the physical volume names explicitly.

   2. The maximum physical volume number.

      You have to specify this number explicitly using the -u option of the `mklv` command or in the SMIT dialog.

   3. Both of the above.

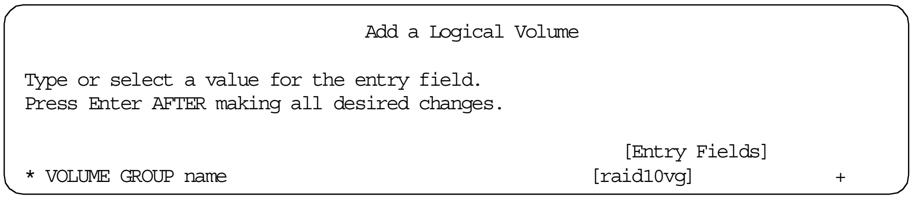
b. The stripe unit size.

   You have to specify this number explicitly using the -S option of the `mklv` command or in the SMIT dialog.

The following example shows this procedure using SMIT.

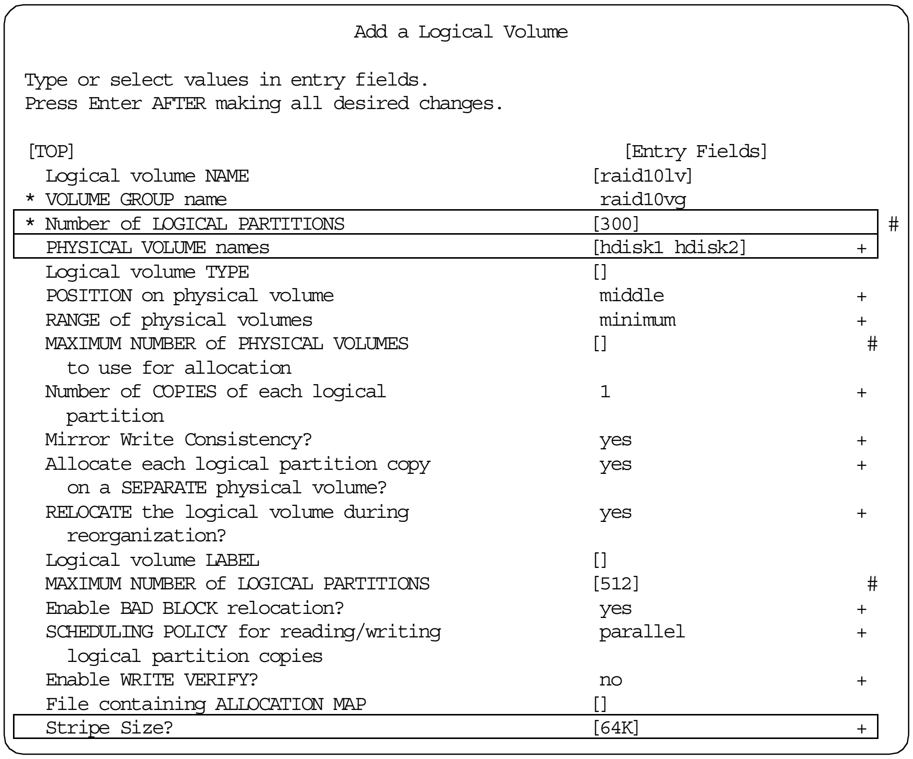
```
root@lorraine:/ [53] # smit lv
    Add a Logical Volume
```

Enter the volume group name in the following dialog.



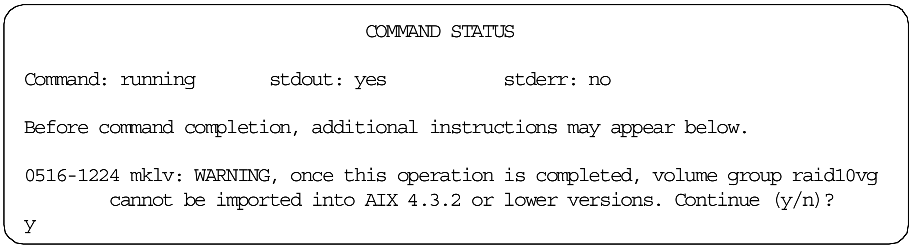
```
                                Add a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* VOLUME GROUP name                                  [raid10vg]           +
```

Input the necessary information. In this example, we specified the information shown in the rectangles in the following SMIT dialog panel screen.



```
                              Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                 [Entry Fields]
  Logical volume NAME                               [raid10lv]
* VOLUME GROUP name                                  raid10vg
* Number of LOGICAL PARTITIONS                      [300]               #
  PHYSICAL VOLUME names                             [hdisk1 hdisk2]    +
  Logical volume TYPE                               []
  POSITION on physical volume                        middle            +
  RANGE of physical volumes                          minimum           +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                []                  #
    to use for allocation
  Number of COPIES of each logical                   1                 +
    partition
  Mirror Write Consistency?                          yes               +
  Allocate each logical partition copy               yes               +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                 yes               +
    reorganization?
  Logical volume LABEL                              []
  MAXIMUM NUMBER of LOGICAL PARTITIONS              [512]               #
  Enable BAD BLOCK relocation?                       yes               +
  SCHEDULING POLICY for reading/writing              parallel          +
    logical partition copies
  Enable WRITE VERIFY?                               no                +
  File containing ALLOCATION MAP                    []
  Stripe Size?                                      [64K]              +
```

Then, you will see the following warning message. Press **y** and **Enter**.



```
                               COMMAND STATUS

Command: running       stdout: yes          stderr: no

Before command completion, additional instructions may appear below.

0516-1224 mklv: WARNING, once this operation is completed, volume group raid10vg
        cannot be imported into AIX 4.3.2 or lower versions. Continue (y/n)?
y
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [378] # mklv -y raid10lv -S 64K raid10vg 300 hdisk1 hdisk2
```

2. Create the mirror of the striped logical volume.

   To create the striped logical volume, you have to explicitly specify the following information:

   a. Where you want to create the mirror

You have to choose one of the following:

1. The physical volume names.

   You have to specify the physical volume name list explicitly.

2. The maximum physical volume number.

   You have to specify this number explicitly using the -u option of the `mklvcopy` command or in the SMIT dialog.

3. Both of the above.

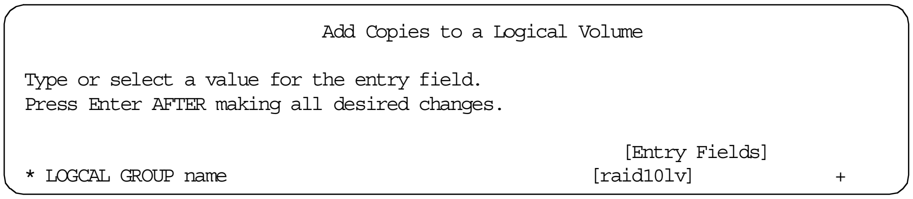
b. The copy number.

   You have to specify this number explicitly using the -c option of the `mklvcopy` command or in the SMIT dialog.

The following example shows this procedure using SMIT.

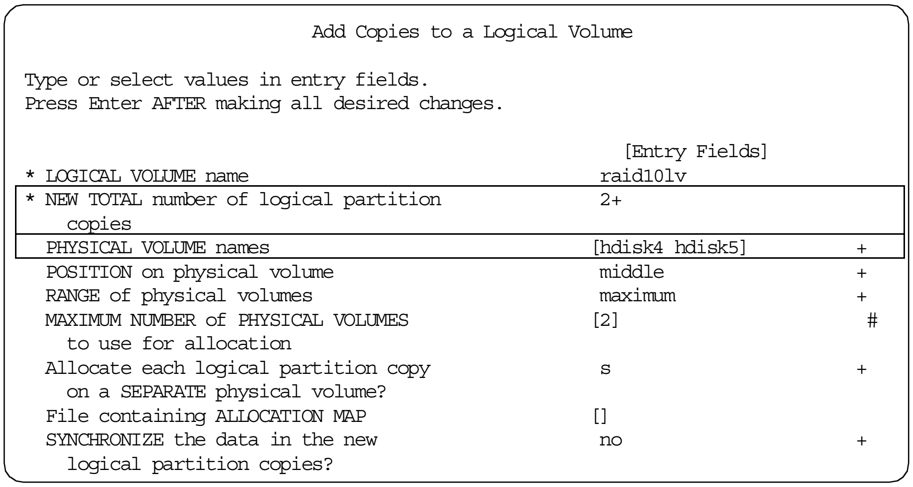
```
root@lorraine:/ [54] # smit lv
    Set Characteristic of a Logical Volume
        Add a Copy to a Logical Volume
```

Enter the logical volume name on the following dialog.



```
                             Add Copies to a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
* LOGCAL GROUP name                                    [raid10lv]            +
```

Input the necessary information. In this example, we specify the information shown in the rectangles in the following SMIT dialog panel screen.



```
                        Add Copies to a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* LOGICAL VOLUME name                                  raid10lv
* NEW TOTAL number of logical partition                2+
    copies
  PHYSICAL VOLUME names                               [hdisk4 hdisk5]          +
  POSITION on physical volume                          middle                  +
  RANGE of physical volumes                            maximum                 +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  [2]                       #
    to use for allocation
  Allocate each logical partition copy                 s                       +
    on a SEPARATE physical volume?
  File containing ALLOCATION MAP                      []
  SYNCHRONIZE the data in the new                      no                      +
    logical partition copies?
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [386] # mklvcopy raid10lv 2 hdisk4 hdisk5
0516-1224 mklvcopy: WARNING, once this operation is completed, volume group raid10vg
        cannot be imported into AIX 4.3.2 or lower versions. Continue (y/n)?
```

3. Examine the physical partitions allocation of the striped and mirrored logical volume.

   To perform this step, you can use the following command option -m of the `lslv` command. It is also graphically represented in Figure 62 on page 174.

```
root@lorraine:/ [388] # lslv -m raid10lv
raid10lv:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0109 hdisk1            0109 hdisk4
0002  0109 hdisk2            0109 hdisk5
0003  0110 hdisk1            0110 hdisk4
0004  0110 hdisk2            0110 hdisk5
(many lines are snipped off)
0297  0257 hdisk1            0257 hdisk4
0298  0257 hdisk2            0257 hdisk5
0299  0258 hdisk1            0258 hdisk4
0300  0258 hdisk2            0258 hdisk5
```

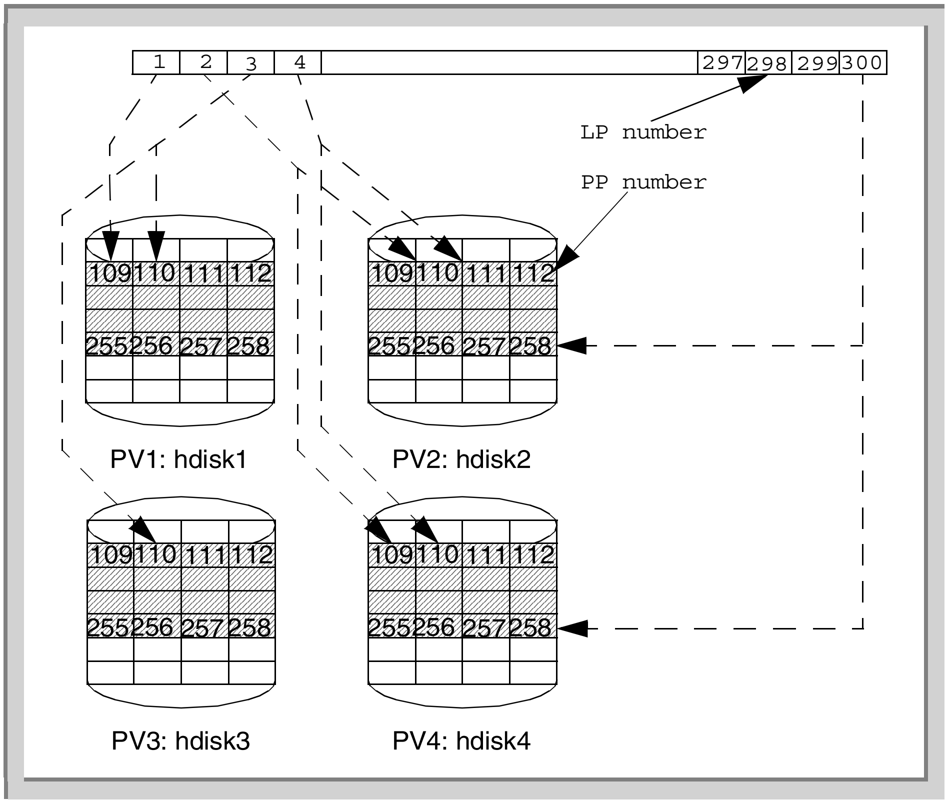


Figure 62. LVs and PVs mapping of a mirrored and striped logical volume

4. Synchronize the mirror

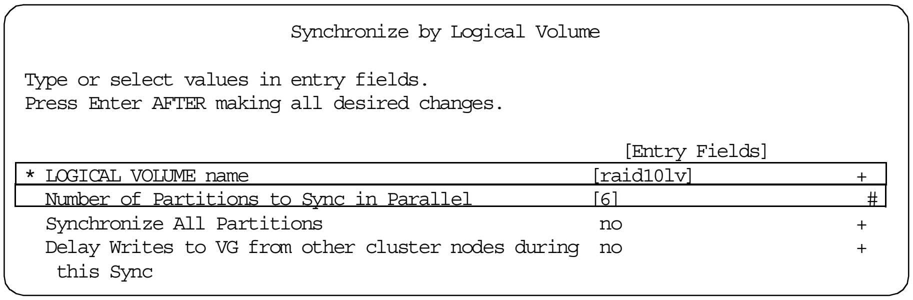
   To synchronize the copies of the specific logical volume, you have to explicitly specify the logical volume name.

   You can specify the concurrency during synchronization (maximum number of partitions synchronize in parallel). To force this, you have to explicitly specify the -P option in the `syncvg` command or in the SMIT dialog.

   The following example shows this procedure using the SMIT.

```
root@lorraine:/ [55] # smit vg
    Synchronize LVM Mirrors
        Synchronize by Logical Volume
```

   Enter the necessary information. In this example, we specify the information shown in the rectangles in the following SMIT dialog.



```
                      Synchronize by Logical Volume

 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* LOGICAL VOLUME name                                 [raid10lv]             +
  Number of Partitions to Sync in Parallel            [6]                     #
  Synchronize All Partitions                           no                    +
  Delay Writes to VG from other cluster nodes during   no                    +
   this Sync
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [410] # syncvg -l -P 6 raid10lv
```

Note

The -P option of the `syncvg` command is introduced in AIX Version 4.3.2. It specifies the concurrency during synchronization to enhance the synchronization speed. The option takes from 1 to 32 as a value. In our experience, the best performance value is six in most circumstances.

Specify to create the mirrors during the initial creation

1. If you would like to specify to create the mirrors of the striped logical volume upon initial creation time, then you have to explicitly specify the following information.

   a. Where you want to create the striped logical volumes

      You have to choose one of the following:

      1. The physical volume names

         You have to specify the physical volume names explicitly.

      2. The maximum physical volume number

         You have to specify this number explicitly using the -u option of the `mklv` command or in the SMIT dialog.

      3. Both of the above

   b. The copy number

      You have to specify this number explicitly using the -c option of the `mklv` command or in the SMIT dialog.

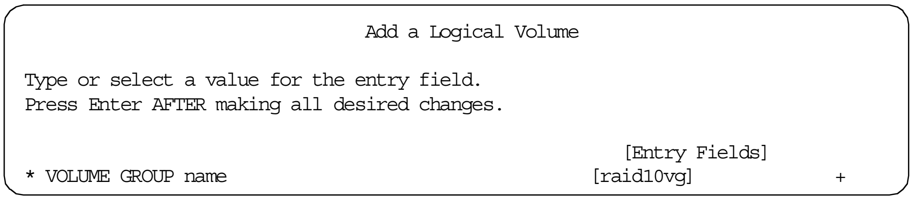
   c. The stripe unit size

      Specify this number explicitly using the -S option of the `mklv` command or in the SMIT dialog.

The following example shows this procedure using SMIT.

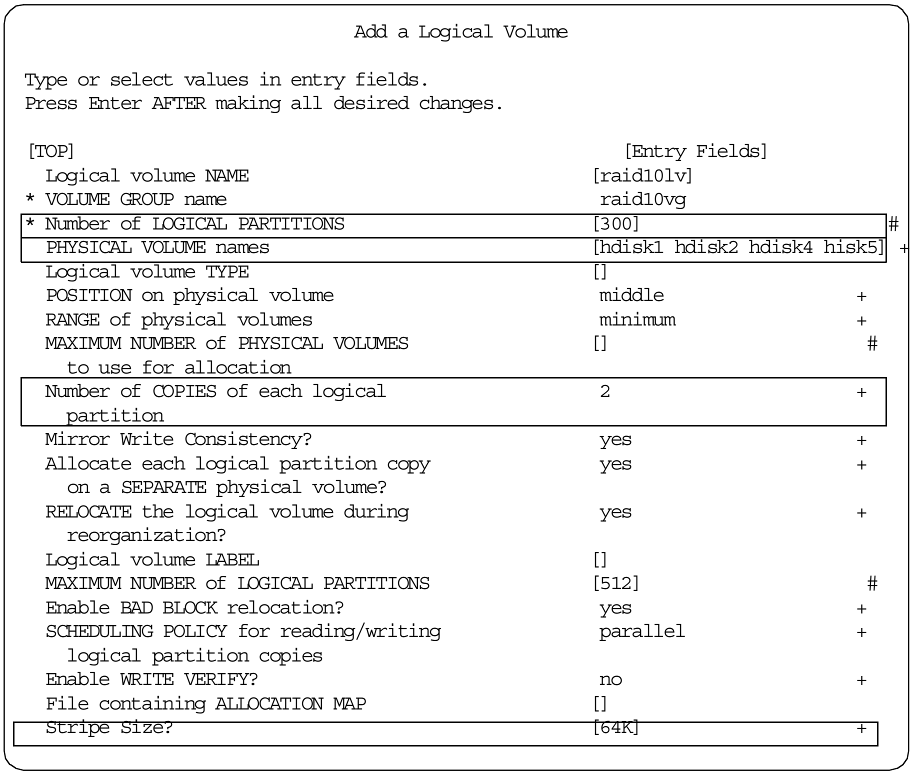
```
root@lorraine:/ [178] # smit lv
    Add a Logical Volume
```

Enter the volume group name in the following dialog.



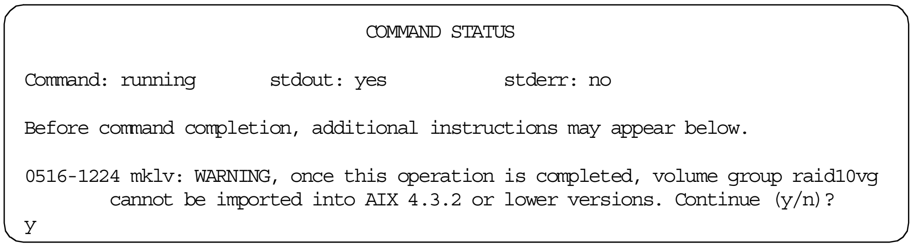
```
                                Add a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* VOLUME GROUP name                                  [raid10vg]            +
```

Input the necessary information. In this example, we specified the information shown in the rectangles in the following SMIT dialog panel screen.



```
                               Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
  Logical volume NAME                                [raid10lv]
* VOLUME GROUP name                                   raid10vg
* Number of LOGICAL PARTITIONS                       [300]                  #
  PHYSICAL VOLUME names                              [hdisk1 hdisk2 hdisk4 hisk5]  +
  Logical volume TYPE                                []
  POSITION on physical volume                         middle               +
  RANGE of physical volumes                           minimum              +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                 []                     #
    to use for allocation
  Number of COPIES of each logical                    2                    +
    partition
  Mirror Write Consistency?                           yes                  +
  Allocate each logical partition copy                yes                  +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                  yes                  +
    reorganization?
  Logical volume LABEL                               []
  MAXIMUM NUMBER of LOGICAL PARTITIONS               [512]                  #
  Enable BAD BLOCK relocation?                        yes                  +
  SCHEDULING POLICY for reading/writing               parallel             +
    logical partition copies
  Enable WRITE VERIFY?                                no                   +
  File containing ALLOCATION MAP                     []
  Stripe Size?                                       [64K]                 +
```

Then, you will see the following warning message. Press **y** and **Enter**.



```
                              COMMAND STATUS

Command: running       stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

0516-1224 mklv: WARNING, once this operation is completed, volume group raid10vg
        cannot be imported into AIX 4.3.2 or lower versions. Continue (y/n)?
y
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [188] # mklv -c 2 -y raid10lv -S 64K raid10vg 300 hdisk1 hdisk2 hdisk4
hdisk5
```

Note

In this case, the `mklv` command synchronizes all the copies automatically. Therefore, you do not have to issue synchronize. In exchange for this ease, you cannot have the concurrency of the synchronization.

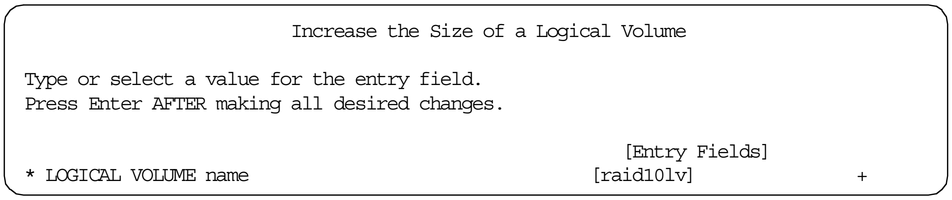
3.3.2.3 Real example of the extension

To extend the striped and mirrored logical volume, you have to explicitly specify how many logical partitions will be extended.

The following example shows this procedure using SMIT.

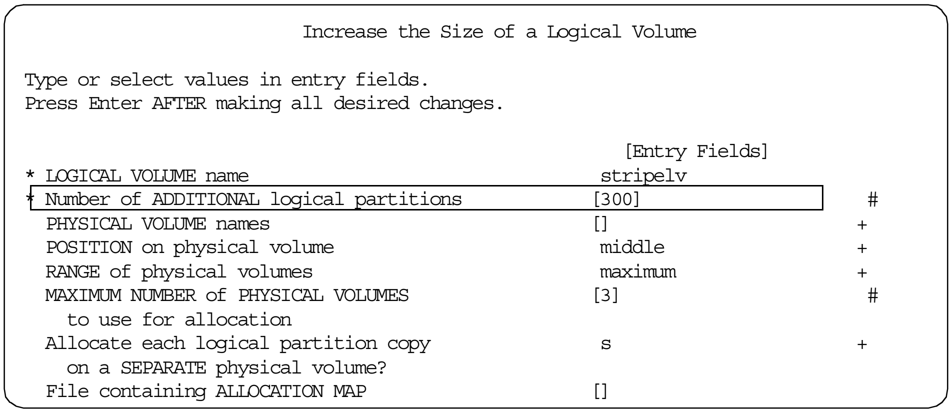
```
root@lorraine:/ [60] # smit lv
    Set Characteristic of a Logical Volume
        Increase the Size of a Logical Volume
```

Enter the logical volume name on the following dialog, then press **Enter**.



```
                        Increase the Size of a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                     [Entry Fields]
* LOGICAL VOLUME name                             [raid10lv]              +
```

Input the necessary information. In this example, we specify the information shown in the rectangles in the following SMIT dialog panel screen.



```
                         Increase the Size of a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
* LOGICAL VOLUME name                                    stripelv
* Number of ADDITIONAL logical partitions               [300]                #
  PHYSICAL VOLUME names                                 []                  +
  POSITION on physical volume                            middle              +
  RANGE of physical volumes                              maximum             +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                    [3]                  #
    to use for allocation
  Allocate each logical partition copy                   s                   +
    on a SEPARATE physical volume?
  File containing ALLOCATION MAP                        []
```

If you prefer the command line interface, you can use the following command:

```
root@lorraine:/ [459] # extendlv raid10lv 100
```

After the extension of this mirrored striped logical volume, you will have the following physical partition allocation (see Figure 63).

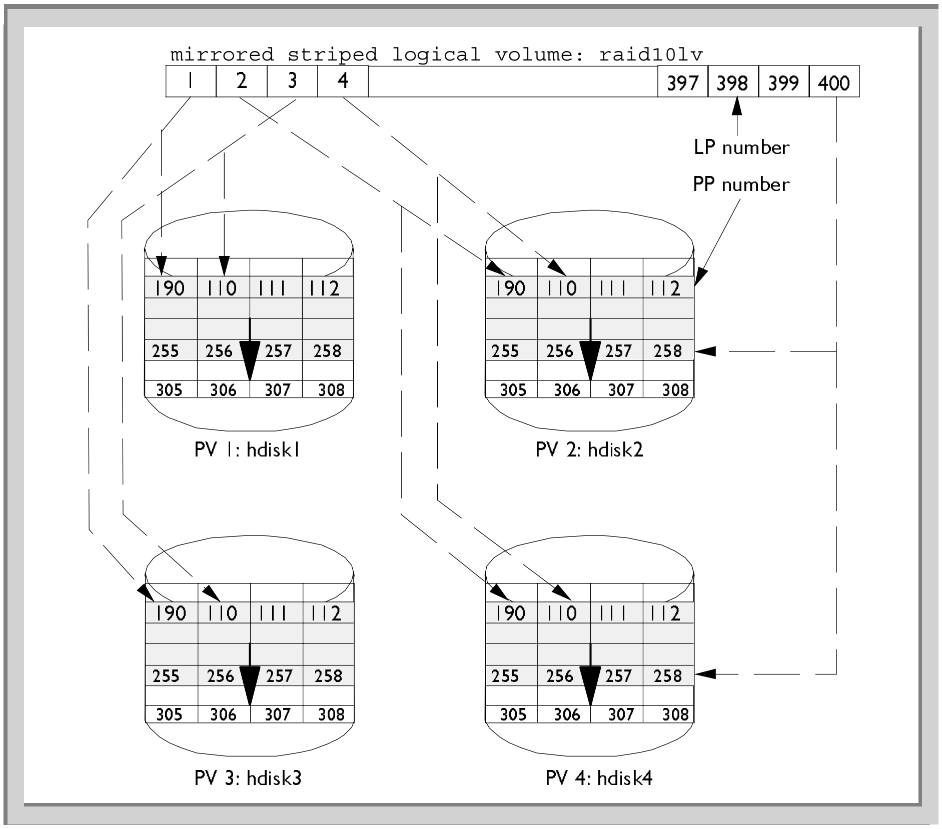


Figure 63. Extension of the mirrored and striped logical volume

3.3.2.4 Real example of the removing of a mirror

To remove one mirror of this mirrored and striped logical volume, you have to explicitly specify the following information.

1. How many copies will be removed

   You have to specify 1 or 2 as the value.

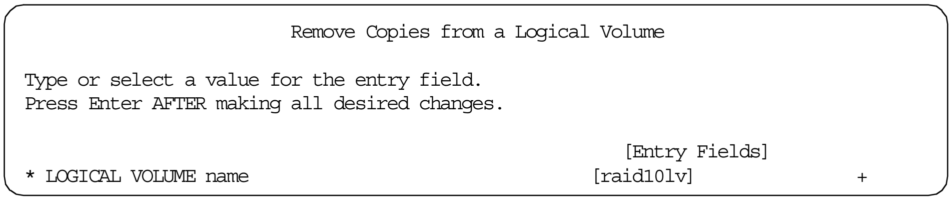
2. From what physical disks the copies will be removed (optional

   You have to specify the physical disk names explicitly.

The following example shows this procedure using SMIT:

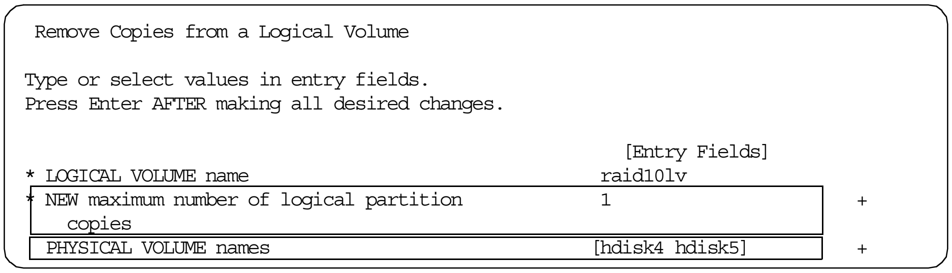
```
root@lorraine:/ [60] # smit lv
    Set Characteristic of a Logical Volume
        Remove a Copy from a Logical Volume
```

Input the logical volume name on the following dialog, then press **Enter**.



```
                         Remove Copies from a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* LOGICAL VOLUME name                                 [raid10lv]            +
```

Input the necessary information. In this example, we specify the information shown in the rectangles in the following SMIT dialog panel screen.



```
 Remove Copies from a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* LOGICAL VOLUME name                                  raid10lv
* NEW maximum number of logical partition              1                    +
    copies
  PHYSICAL VOLUME names                               [hdisk4 hdisk5]       +
```

If you prefer the command line interface, you can use the following command line:

```
root@lorraine:/ [462] # rmlvcopy raid10lv 1 hdisk4 hdisk5
```

After removing the mirror from this mirrored and striped logical volume, you will have the following physical partition allocation (see Figure 64 on page 181).

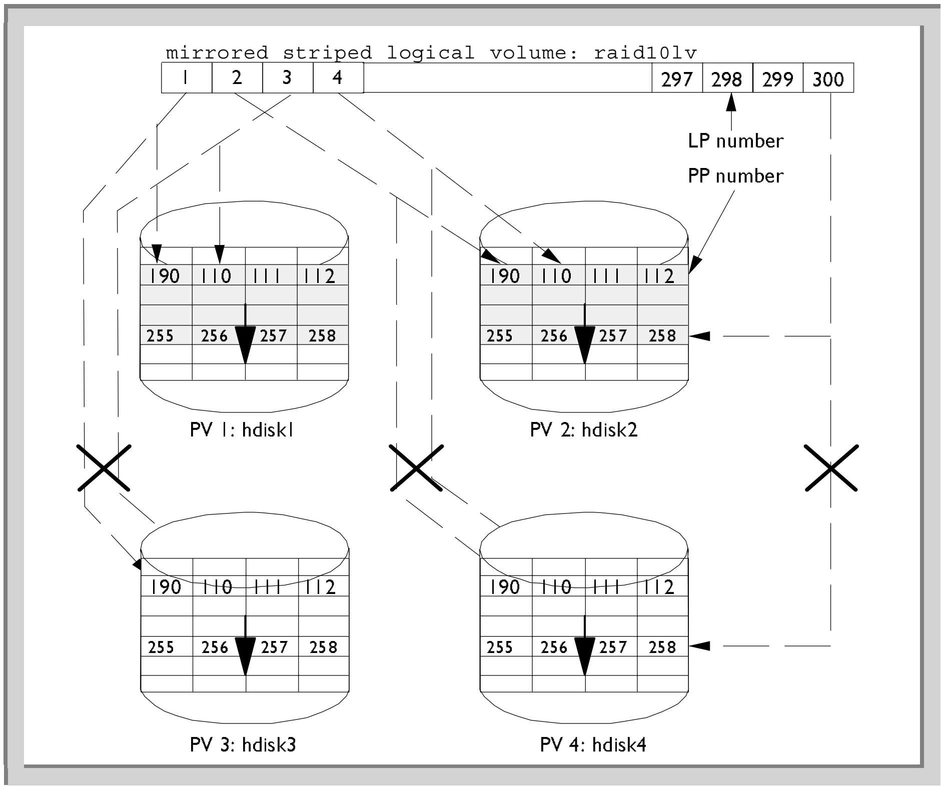


Figure 64. Removal of one mirror from the mirrored and striped logical volume

Chapter 4. Concurrent access volume groups

This chapter will discuss concurrent access volume groups and the related software product from the concepts, real examples, shared disk environment solutions, and the history of concurrent access.

4.1 Concept

The concurrent access volume group is a volume group that can be accessed from more than one host system simultaneously (therefore it is called *concurrent access*). This capability is achieved by applying the following concepts and functional elements.

4.1.1 Basic schema of the concurrent access

Usually, in non-concurrent access mode, volume groups can be varied online from only one host system. While one host system can access the physical volumes, these are put into a reserved status. This means the physical volumes are used by only one disk adapter. The reserved status disk drives refuse the order issued from any other disk adapter (see Figure 65 on page 183).

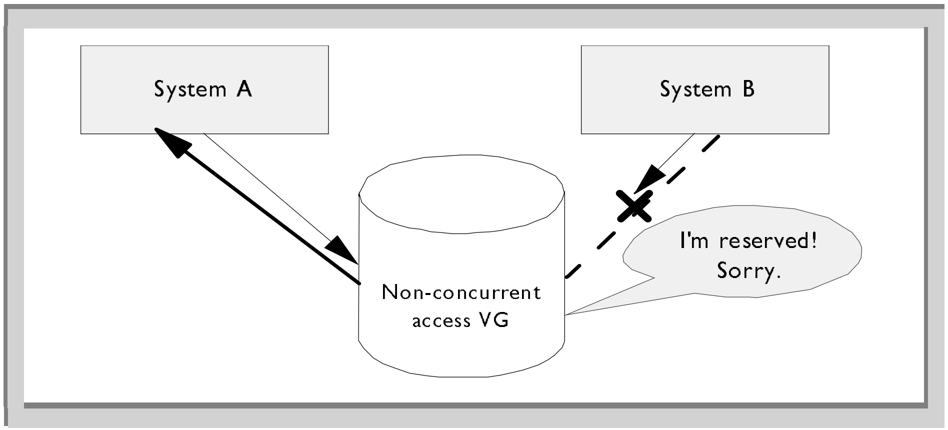


Figure 65. Varied online with non-concurrent mode

In other words, the disk drives that compose the non-concurrent access mode volume groups are put into reserved status permanently until varied offline.

Note

In SCSI terminology, the adapters that initiates the I/O operation to the disk drives are called *initiators*, and the disk drives are called *targets*.

In concurrent access mode, the disk drives are not put into this reserved state. The disk drives are not protected from any order from other adapters. Any systems that share the same external I/O buses (usually, SCSI or SSA buses), can send any I/O request to that disk drive (see Figure 66). Of course, since actual physical disk drives are not multi-task capable, there is always a possibility that the other system is using the physical disk drive at a particular instant. The I/O requests that are issued later (shown as the arrow with the check mark in Figure 66), could be suspended, blocked, or even worse returned with time-out. The actual behavior depends on the hardware related timing issue. Hence, in the concurrent access environment, the supported hardware are limited as described in 4.1.4, “Limitations” on page 193.

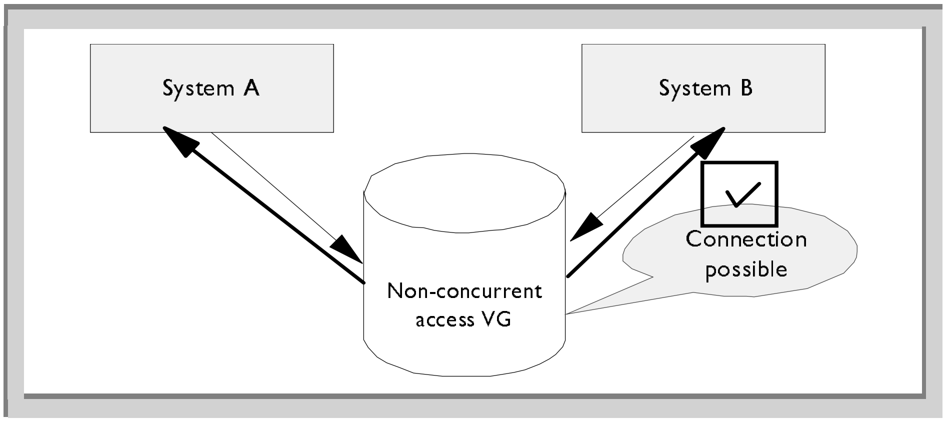


Figure 66. Varied online with concurrent mode

Generally speaking, from the LVM perspective, there are two categories of access models to the volume groups (including the concurrent access mode), although there are no distinct differences in the hardware level I/O request.

- LVM configuration operations
- I/O operations to the logical volumes

In the concurrent access volume groups, the first category of operations is under the control of the concurrent logical volume manager (CLVM). For detailed information about the CLVM, please refer to 4.1.3, “Concurrent Logical Volume Manager (CLVM)” on page 187.

On the other hand, the second category of operations is not ruled by the AIX LVM on the concurrent access volume groups. It is completely up to the applications that access concurrent access volume groups to control who does what with the logical volumes. The lack of this control could immediately result in the data divergence problems among the systems that are sharing the concurrent access volume groups. For further detailed information about the HACMP cluster lock manager, please refer to 4.3.2, “Cluster lock manager” on page 223.

4.1.2 Concurrent access capable and concurrent mode

Here is the first element to understand the concurrent access volume group: There are two different attributes to designate the behavior of the concurrent access volume group. They are called *concurrent access capable* and *concurrent mode*. These are attributes of the volume group, but never show up in normal (non-concurrent access) volume groups. See the following two examples.

- Case of a normal volume group:

```
root@goofy:/ # lsvg rootvg
VOLUME GROUP:   rootvg                   VG IDENTIFIER:  0000107319fd31ee
VG STATE:       active                   PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      159 (636 megabytes)
MAX LVs:        256                      FREE PPs:       2 (8 megabytes)
LVs:            8                        USED PPs:       157 (628 megabytes)
OPEN LVs:       7                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     1                        AUTO ON:        yes
MAX PPs per PV: 1016                     MAX PVs:        32
```

- Case of a concurrent access volume group:

  The attributes shown in the rectangle in the following example are only displayed for a concurrent access volume group. In this case, this volume group is created as concurrent access capable and varied online with non-concurrent mode.

  - Concurrent access capable, with non-concurrent mode

```
root@goofy:/ # lsvg concvg
VOLUME GROUP:   concvg                   VG IDENTIFIER:  000010736127caf7
VG STATE:       active                   PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      203 (812 megabytes)
MAX LVs:        256                      FREE PPs:       152 (608 megabytes)
LVs:            2                        USED PPs:       51 (204 megabytes)
OPEN LVs:       0                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     1                        AUTO ON:        no
Concurrent:     Capable                  Auto-Concurrent: Disabled
VG Mode:        Non-Concurrent
Node ID:        1                        Active Nodes:   2
MAX PPs per PV: 1016                     MAX PVs:        32
```

But, the concurrent access volume group can be varied online with concurrent mode enabled:

- Concurrent access capable, with concurrent mode

```
root@goofy:/ # lsvg concvg
VOLUME GROUP:   concvg                   VG IDENTIFIER:  000010736127caf7
VG STATE:       active                   PP SIZE:        4 megabyte(s)
VG PERMISSION:  read/write               TOTAL PPs:      203 (812 megabytes)
MAX LVs:        256                      FREE PPs:       152 (608 megabytes)
LVs:            2                        USED PPs:       51 (204 megabytes)
OPEN LVs:       0                        QUORUM:         2
TOTAL PVs:      1                        VG DESCRIPTORS: 2
STALE PVs:      0                        STALE PPs:      0
ACTIVE PVs:     1                        AUTO ON:        no
Concurrent:     Capable                  Auto-Concurrent: Disabled
VG Mode:        Concurrent
Node ID:        1                        Active Nodes:   2
MAX PPs per PV: 1016                     MAX PVs:        32
```

The concurrent access volume groups can be accessed from multiple nodes concurrently, if they are brought online with concurrent mode. If the concurrent access volume groups are varied online with non-concurrent mode, then only the node that issued the `varyonvg` command can access it.

To summarize these two attributes, we use the following table.

Table 14. Concurrent access capable and concurrent mode

	Normal volume groups	**Concurrent access volume groups**
Concurrent access capable	No (never displayed)	Yes

	Normal volume groups	**Concurrent access volume groups**
Concurrent mode	No (never displayed)	Concurrent or Non-Concurrent

4.1.3 Concurrent Logical Volume Manager (CLVM)

The concurrent logical volume manager (CLVM) is a daemon subsystem that is a part of the fileset bos.rte.lvm (after AIX Version 4.2. Please refer to 4.4, “History of the concurrent access” on page 231). It has the following file path name:

```
root@mickey:/ # whence clvmd
/usr/sbin/clvmd
root@mickey:/ # lslpp -w /usr/sbin/clvmd
  File                                        Fileset               Type
  ----------------------------------------------------------------------------
  /usr/sbin/clvmd                             bos.rte.lvm           File
```

and the following ODM definition:

```
root@mickey:/ # odmget -q subsysname=clvmd SRCsubsys

SRCsubsys:
        subsysname = "clvmd"
        synonym = ""
        cmdargs = ""
        path = "/usr/sbin/clvmd"
        uid = 0
        auditid = 0
        standin = "/dev/console"
        standout = "/dev/console"
        standerr = "/dev/console"
        action = 2
        multi = 0
        contact = 2
        svrkey = 0
        svrmtype = 0
        priority = 20
        signorm = 15
        sigforce = 15
        display = 1
        waittime = 20
        grpname = ""
```

The CLVM has the responsibility for the following LVM configuration operations on the concurrent access volume groups:

- Device configuration operation through the LVM device driver (varyon with concurrent mode).
- Operations that require ODM updates:
  - Adding and deleting the logical volumes
  - Extending and reducing the logical volume sizes

- Changing attributes of the logical volume
- Adding and removing copies of the mirrored logical volumes

4.1.3.1 Vary on with concurrent mode

To vary on the concurrent access volume group with concurrent mode, you must have the following prerequisite conditions in addition to the CLVM software installed.

- Supported physical shared disk configurations:

  This condition is required by the HACMP/CRM product (High Availability Cluster Multi-Processing for AIX, Concurrent Resource Manager feature). For further detailed information about this product, please refer 4.3.1, “HACMP and HACMP Enhanced Scalability (ES)” on page 219. Note that we use the word HACMP to express both the HACMP and the HACMP/ES products through this chapter.

- Installation of HACMP/CRM.

  This installation is necessary for enabling the CLVM to have the ability to vary on the concurrent access volume groups with concurrent mode. For further detailed information about this installation, please refer to 4.2.3, “Installing the HACMP concurrent resource manager (CRM)” on page 200.

- Necessary HACMP definition for the concurrent access volume group (called *concurrent resource group*). For further detailed information about this definition, please refer to 4.2.4, “Installing HACMP cluster configurations: Standard method” on page 203.

The last prerequisite condition is officially documented in the command reference of the `mkvg` and `varyonvg` command. The following descriptions are excerpts from these command references.

- `mkvg`

  -c Creates a Concurrent Capable volume group. Only use the -c flag with the HACMP. It has no effect on volume groups and systems not using the HACMP product. This flag only applies to AIX Version 4.2 or later.

- `varyonvg`

  -c Varies the volume group on in concurrent mode. This is only possible if the volume group is Concurrent Capable and the system has the HACMP product loaded and available. If neither is true, the volume group will fail the varyon. This flag only applies to AIX Version 4.2 or later. If the varyon process detects that there is a new logical volume in the volume group whose name is already being used for one of the existing logical volumes, then the varyon will fail. You will need to rename the existing logical volume before attempting the varyon again.

Note

HACMP should always be configured to vary on concurrent volume groups. Under normal runtime conditions, concurrent volume groups should never be started at boot time or by hand.

Suppose that there are three nodes that share the same external I/O bus, and only two nodes (represented as the System A and the System B in Figure 67) have the necessary HACMP definition (represented as the small black rectangles). In this case, the System C, which does not have the necessary HACMP definition, can not vary online this volume group in concurrent mode.

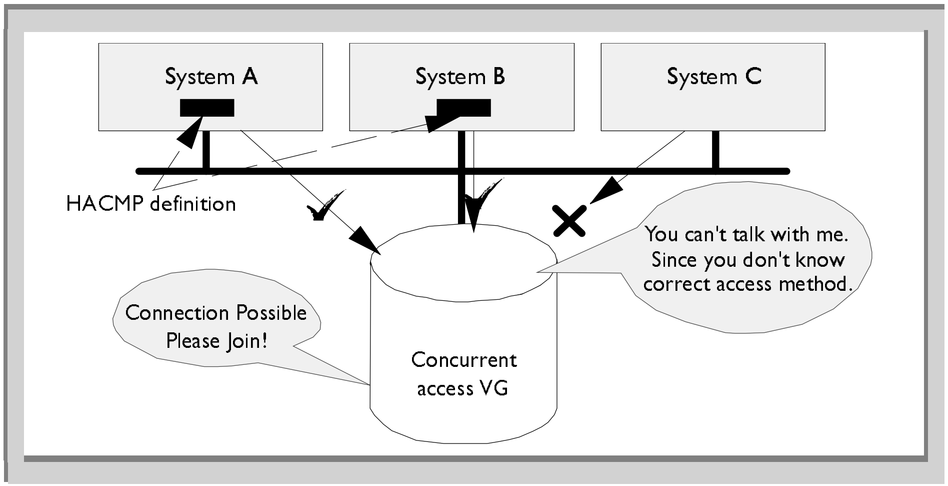

System A
System B
System C
HACMP definition
You can't talk with me. Since you don't know correct access method.
Connection Possible Please Join!
Concurrent access VG

Figure 67. One system does not have any ODM definition

Note that the ability of the CLVM which prevents the unnecessary I/O access is based on the LVM. If you bypass the LVM to access to the physical volumes that compose the concurrent access volume group (for example, access the /dev/hdiskX block device special file directly), there is no protection mechanism provided anymore.

Usually, the concurrent access volume group is varied online by the HACMP software upon the cluster manager startup using the cl_mode3 script which is called in the node_up_local event. So, it is not necessary to vary online the concurrent access volume group manually, except for the troubleshooting purpose in the initial setup and the problem determination phase.

For example, if you issue the `varyonvg` command using the -c option to vary online the concurrent access volume group with concurrent mode manually, then you will see the following startup message and find out that the `clvmd` is automatically invoked.

```
root@goofy:/ # varyonvg -c concvg
0513-059 The clvmd Subsystem has been started. Subsystem PID is 13530.
root@goofy:/ # ps -ef | grep [c]lvmd
    root 13530  3646   0 22:01:42      -  0:00 /usr/sbin/clvmd
```

The clvmd can be restarted using the `varyonvg -c` command or the `startsrc -s clvmd` command.

4.1.3.2 Vary on with concurrent mode without prerequisites

If you create the concurrent access volume group and just attempt to vary it online with concurrent mode forcibly without installing the HACMP/CRM, you will see the following error message. The concurrent mode is forced using the -c option of the `varyonvg` command.

```
root@goofy:/ # varyonvg -c concvg
0516-945 varyonvg: The library /usr/lib/libclstr.a is not found on the system.
Concurrent operations not allowed.
0516-008 varyonvg: LVM system call returned an unknown error code (-1).
```

You can not vary it online with concurrent mode. The library archive file is a part of the fileset named cluster.base.client.lib which is included in the HACMP product.

```
root@mickey:/ # lslpp -w /usr/lib/libclstr.a
  File                                          Fileset               Type
  ----------------------------------------------------------------------------
  /usr/lib/libclstr.a
                                     cluster.base.client.lib       File
```

Installing this fileset is still not enough. You have to create the necessary definition with HACMP (called *concurrent resource group*). If you tried to bring this volume group online without this definition, you will see the following message.

```
root@goofy:/ # lslpp -L cluster.base.client.rte
cluster.base.client.lib    4.3.1.0    C    HACMP Base Client Libraries.
root@goofy:/ # ls -l /usr/lib/libclstr.a
-r--r--r--   1 bin      system    140949 Jun 15 11:57 /usr/lib/libclstr.a
root@mickey:/ # varyonvg -c concvg
0513-059 The clvmd Subsystem has been started. Subsystem PID is 12680.
PV Status:      hdisk1  0000098547779f9a        PVACTIVE
0516-078 varyonvg: Incomplete device driver configuration. Probable
        cause is the special file for the VG is missing. Execute
        varyoffvg followed by varyonvg to build correct environment.
```

To fix this erroneous situation, you have to issue the following commands:

```
root@mickey:/ # varyonvg concvg
root@mickey:/ # varyoffvg concvg
```

In our experience with this test, the `varyonvg` and `varyoffvg` commands are not enough to fix this situation. We have to issue the `exportvg` command and re-create the volume group.

4.1.3.3 LVM configuration operations that require ODM updates

These operations include the following tasks and require ODM updates on each node that shares the concurrent access volume groups.

- Adding and deleting the logical volumes
- Extending and reducing the logical volume sizes
- Changing some attributes of the logical volume
- Adding and removing the copy of the mirrored logical volumes

To enable these ODM updates on each node, two essential components, the CLVM and the concurrent configuration scratch record, exist. The CLVM is a daemon subsystem named clvmd that is invoked automatically upon the vary on of the concurrent access volume group on each machine that shares the same external I/O bus.

The concurrent configuration scratch record is eight sectors long and is located in the first 128 sectors of the LVM reserved area. This reserved area is located at the very beginning of each physical disk drive. The concurrent configuration scratch records are defined in the /usr/include/sys/hd_psn.h header file (shown in the rectangle in the following example).

```
/*
 *   This file contains the physical sector number (PSN) layout of
 *   reserved space on the hardfile.
 */
(some lines are snipped out)
#define  PSN BB BAK      71    /* PSN of backup bad block directory     */
#define  PSN_CFG_TMP     120   /* PSN of concurrent config work record  */
#define  PSN_NONRSRVD    128   /* PSN of first non-reserved sector      */

/*
 * The concurrent config scratch sectors (120-128)
 * are accessible thru LV0 as a negative offset from
 * the VGSA area, which starts at PSN_NONRSRVD.
 * Note that there is one config scratch area
 * per pvol.  When a config operation is sent across,
 * We need to write each of them in turn.
 * When reading them, we stop once we hit a match--
 * (one which contains the same OP/NODEID/HINT as the config message)
 */
#define  GETCFG_LSN(Vg, copy)   \
                (GETSA_LSN(Vg, copy<<1) - (PSN_NONRSRVD - PSN_CFG_TMP))
```

If you attempt the LVM configuration operations that require ODM updates (for example making the logical volume) on one system (called *initiator node*), the instruction is then captured by the clvmd on that system.

If the `clvmd` on the other system (called *passive node*) is notified that there is a LVM configuration operation attempt, it also read additional information along with the operation message on the concurrent configuration scratch record. Note that some hardware like the 9333 disk subsystem has a special hardware register mechanism that is used for notification in this communication process.

The `clvmd` performs the processing necessary to make the passive node's view of the shared volume group consistent with the initiator node's view, including updating the in-kernel copies of data structures related to the volume group.

The ODM definition on the passive node is also updated as part of this step. When this has been successfully completed, the passive node sends an acknowledgement message back to the initiator node through the same communication method. The progress of this update procedure can be tracked on all nodes by monitoring the /tmp/clvmd.log file as shown below:

```
BEGIN:
VGID:1073 6127caf7 0 0
dev: 144000 (20/16384)
vgmajor = 51, op=62 node=1
Config op 62 (HD_CONFIG_TAKECONCH)
lvm_config(16958092, 0, HD_CONFIG_TAKECONCH) returns 0
```

```
BEGIN:
VGID:1073 6127caf7 0 0
dev: 144000 (20/16384)
vgmajor = 51, op=80 node=1
Default config op 80
config op 80 succeeded
lvm_config(16958092, 0, HD_CONFIG_ACK) returns 0


BEGIN:
VGID:1073 6127caf7 0 0
dev: 144000 (20/16384)
vgmajor = 51, op=20 node=1
Config op 20 KADDLV SUCCESSFUL
Updating ODM...lvm_config(16958092, 0, HD_CONFIG_ACK) returns 0


BEGIN:
VGID:1073 6127caf7 0 0
dev: 144000 (20/16384)
vgmajor = 51, op=30 node=1
Config op 30 KEXTEND
VGID: 1073/6127caf7/0/0
lv_minor = 2
copies = 1
num lps = 50
num extred = 50

lp_num = 1, pv_num = 1, start_addr = 348416, mask = 0x0
lp_num = 2, pv_num = 1, start_addr = 356608, mask = 0x0
lp_num = 3, pv_num = 1, start_addr = 364800, mask = 0x0
lp_num = 4, pv_num = 1, start_addr = 372992, mask = 0x0
lp_num = 5, pv_num = 1, start_addr = 381184, mask = 0x0
lp_num = 6, pv_num = 1, start_addr = 389376, mask = 0x0
lp_num = 7, pv_num = 1, start_addr = 397568, mask = 0x0
lp_num = 8, pv_num = 1, start_addr = 405760, mask = 0x0
lp_num = 9, pv_num = 1, start_addr = 413952, mask = 0x0
lp_num = 10, pv_num = 1, start_addr = 422144, mask = 0x0
```

4.1.4 Limitations

The use of the concurrent access volume groups has some limitation listed as follow:

- The hardware support is limited.

  From the *HACMP Version 4.3 AIX: Installation Guide,* SC23-4278, only the following external disk subsystems are supported by the HACMP/CRM:

  - IBM 7135-110 and 210 RAIDiant disk array
  - IBM 7137 disk array
  - IBM 9333 serial disk subsystem
  - IBM 7133 or 7131-405 SSA disk subsystem
  - IBM 2105-B09 and 100 Versatile Storage Server

Note

The SCSI RAID systems do not offer online concurrent volume group management.Those systems are not eligible either for the `mirrorvg` command.

For new hardware support information, please consult the appropriate HACMP service update information.

- Big VGDA format is not supported on the concurrent volume group.

  In the big VGDA format volume groups, the additional information to update the ODM on passive nodes is bigger than the normal VGDA format volume groups. Therefore, the big VGDA format is not supported by the concurrent access on the current implementation.

- JFS is not supported on the concurrent access volume groups.

  All the UNIX implementations, including the AIX one, have the super block cached in memory to accelerate access to the file system created on the physical disk drive. There is no way to synchronize this super block cache on each node that shares the concurrent access volume group. Hence, JFS is not supported in the concurrent access volume groups.

- MWCC should be disabled on the concurrent access volume group.

  The above explanation about the super block cache is also valid for the MWCC (Mirror Write Consistency Checking) cache held in the memory on each node that shares the concurrent access volume group. Therefore, the MWCC should be disabled with the logical volume in concurrent access volume group. It also implies that if all the cluster nodes failed at the same time, there is no way to determine which copy is the correct one.

- Bad block relocation should be disabled on the concurrent access volume groups.

  The bad block relocation falls in the same category. Since the bad block relocation record is also kept as a cache in the LVM device driver's own memory area on each node, there is no way to prohibit the LVM device driver on each node from overwriting the bad block relocation records used by other nodes. Therefore, the bad block relocation should be disabled for every logical volume on the concurrent access volume group as well.

Note

The last two limitations are not described (or at least not well explained) in most AIX publications, even in the HACMP Installation Guide and Administration Guide. Keep these limitations in mind.

4.2 Real examples

This section provides real examples about the following tasks:

- Creating the concurrent access volume groups
- Defining the concurrent resource group on HACMP
- Managing the shared LVM components for concurrent access

Actually, these tasks are organized to conform with the HACMP installation steps which are explained indepth in the *HACMP Version 4.3 AIX: Installation Guide,* SC23-4278.

4.2.1 Example configuration

All the examples shown in this section are tested on the following configuration. This configuration consists of two RS/6000 model 520s. Each node has its own rootvg (hdisk0), and shares three external 9333 serial link disk subsystems (shown as hdisk1, hdisk2, hdisk3 in Figure 68 on page 196). This section provides the results that are detailed in the following chapters of *HACMP Version 4.3 AIX: Installation Guide,* SC23-4278.

- Chapter 4, “Configuring Networks”
- Chapter 5, “Installing Shared Disk Devices”

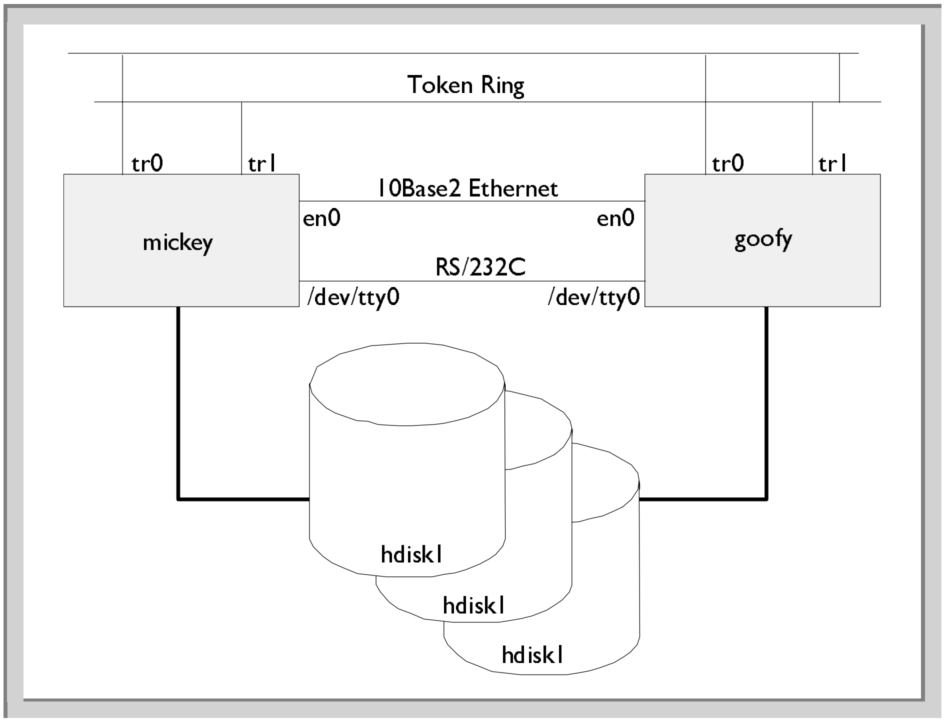


Figure 68. The mickey and goofy configuration

Figure 68 on page 196 presents the connections between the two machines of our cluster. These nodes have the following devices configured.

- Machine A: mickey:

```
root@mickey:/ # lsdev -Cc adapter
(many lines are snipped out)
sa0       Available 00-00-S1 Standard I/O Serial Port 1
sa1       Available 00-00-S2 Standard I/O Serial Port 2
ent0      Available 00-01    Ethernet High-Performance LAN Adapter (8ef5)
tok0      Available 00-02    Token-Ring High-Performance Adapter (8fc8)
tok1      Available 00-04    Token-Ring High-Performance Adapter (8fc8)
serdasda0 Available 00-05    Serial-Link Disk Adapter
scsi0     Available 00-08    SCSI I/O Controller
serdasdc0 Defined   00-05-01 Serial-Link Four Port Controller
serdasdc1 Available 00-05-02 Serial-Link Four Port Controller
root@mickey:/ # lsdev -Cc disk
hdisk0 Available 00-08-00-0,0 1.0 GB SCSI Disk Drive
hdisk1 Available 00-05-02-01  857MB Serial-Link Disk Drive
hdisk2 Available 00-05-02-02  2.0GB Serial-Link Disk Drive
hdisk3 Available 00-05-02-03  2.0GB Serial-Link Disk Drive
root@mickey:/ # lspv
hdisk0         000147320d630b6e    rootvg
hdisk1         0000098547779f9a    None
hdisk2         000009854777a091    None
hdisk3         000009854777a5c6    None
```

```
root@mickey:/ # lsdev -Cc if
en0 Available  Standard Ethernet Network Interface
et0 Defined    IEEE 802.3 Ethernet Network Interface
lo0 Available  Loopback Network Interface
tr0 Available  Token Ring Network Interface
tr1 Available  Token Ring Network Interface
root@mickey:/ # netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 link#1                          2955     0     2990     0     0
lo0   16896 127         127.0.0.1           2955     0     2990     0     0
lo0   16896 ::1                             2955     0     2990     0     0
en0   1500  link#2      2.60.8c.2f.46.db    199269     0   199644     0     0
en0   1500  100.100.100 100.100.100.1       199269     0   199644     0     0
tr0   1492  link#3      10.0.5a.a8.b4.84    295171     0   196023     0     0
tr0   1492  9.3.187.128 9.3.187.184         295171     0   196023     0     0
tr1   1492  link#4      10.0.5a.a8.3c.35    301283     0   172607     0     0
tr1   1492  9.3.187.192 9.3.187.251         301283     0   172607     0     0
root@mickey:/ # lsdev -Cc tty
tty0 Available 00-00-S1-00 Asynchronous Terminal
tty1 Available 00-00-S2-00 Asynchronous Terminal
```

- Machine: goofy:

```
root@goofy:/ # lsdev -Cc adapter
(many lines are snipped out)
sa0       Available 00-00-S1 Standard I/O Serial Port 1
sa1       Available 00-00-S2 Standard I/O Serial Port 2
serdasda0 Available 00-06    Serial-Link Disk Adapter
scsi0     Available 00-08    SCSI I/O Controller
ent0      Available 00-02    Ethernet High-Performance LAN Adapter (8ef5)
tok0      Available 00-01    Token-Ring High-Performance Adapter (8fc8)
tok1      Available 00-03    Token-Ring High-Performance Adapter (8fc8)
serdasdc0 Available 00-06-02 Serial-Link Four Port Controller
root@goofy:/ # lsdev -Cc disk
hdisk0 Available 00-08-00-0,0 670 MB SCSI Disk Drive
hdisk1 Available 00-06-02-01  857MB Serial-Link Disk Drive
hdisk2 Available 00-06-02-02  2.0GB Serial-Link Disk Drive
hdisk3 Available 00-06-02-03  2.0GB Serial-Link Disk Drive
hdisk0         000122180006c324    rootvg
hdisk1         0000098547779f9a    None
hdisk2         000009854777a091    None
hdisk3         000009854777a5c6    None
root@goofy:/ # lsdev -Cc if
en0 Defined    Standard Ethernet Network Interface
et0 Defined    IEEE 802.3 Ethernet Network Interface
lo0 Available  Loopback Network Interface
tr0 Available  Token Ring Network Interface
tr1 Defined    Token Ring Network Interface
root@goofy:/ # netstat -in
Name  Mtu   Network     Address            Ipkts Ierrs    Opkts Oerrs  Coll
lo0   16896 link#1                         85636     0    85670     0     0
lo0   16896 127         127.0.0.1          85636     0    85670     0     0
lo0   16896 ::1                            85636     0    85670     0     0
en0   1500  link#2      2.60.8c.2f.47.42    199642     0   199269     0     0
en0   1500  100.100.100 100.100.100.2       199642     0   199269     0     0
tr0   1492  link#3      10.0.5a.a8.3b.fa    295082     0   189004     0     0
tr0   1492  9.3.187.128 9.3.187.186         295082     0   189004     0     0
tr1   1492  link#4      10.0.5a.a8.d1.f3    293320     0   172268     0     0
tr1   1492  9.3.187.192 9.3.187.252         293320     0   172268     0     0
root@goofy:/ # lsdev -Cc tty
tty0 Available 00-00-S1-00 Asynchronous Terminal
tty1 Available 00-00-S2-00 Asynchronous Terminal
```

Both nodes have the following local host table, and are already assigned the IP addresses shown in Figure 69 to conform with the HACMP IP addressing rule. Please note that the service, boot and standby addresses of the token ring segment have 255.255.255.192 as subnet mask.

- The /etc/hosts file

```
# /etc/hosts -- local host table.
#
127.0.0.1       loopback localhost  # loopback (lo0) name/address
#
# Node: mickey
#
100.100.100.1   mickey_en0  mickey  # host alias trick.
# netmask: 255.255.255.192
9.3.187.183     mickey      mickey.itsc.austin.ibm.com
9.3.187.184     mickey_boot mickey_boot.itsc.austin.ibm.com
9.3.187.251     mickey_stby mickey_stby.itsc.austin.ibm.com
#
# Node: goofy
#
100.100.100.2   goofy_en0 goofy     # host alias trick.
# netmask: 255.255.255.192
9.3.187.185     goofy       goofy.itsc.austin.ibm.com
9.3.187.186     goofy_boot  goofy_boot.itsc.austin.ibm.com
9.3.187.252     goofy_stby  goofy_stby.itsc.austin.ibm.com
#
# end of /etc/hosts -- file was not truncated.
#
```

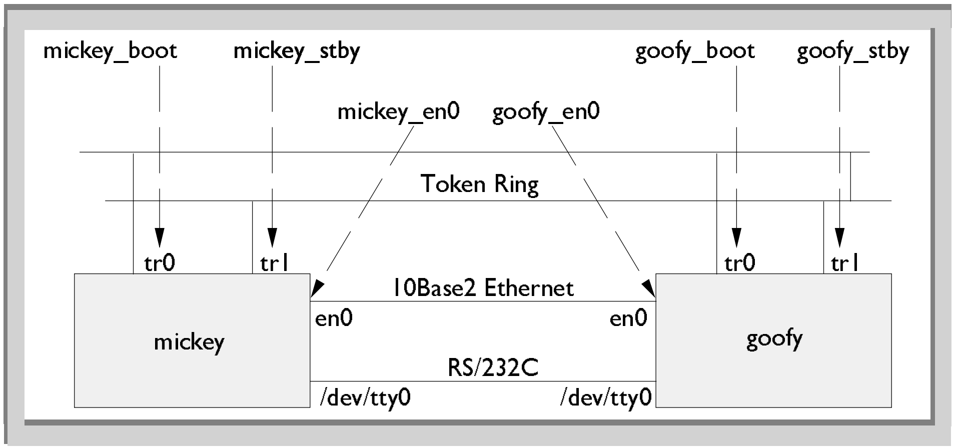


Figure 69. IP address assignment

4.2.2 Defining shared LVM components for concurrent access

This section shows how to create the concurrent access volume group and how to vary it online manually. These tasks are prerequisites for the actual HACMP configuration steps. This section provides the results that are explained in Chapter 6 of the *HACMP Version 4.3 AIX: Installation Guide,* SC23-4278.

4.2.2.1 Creating a concurrent access volume group

To create a concurrent access volume group, you can use the following SMIT panel:

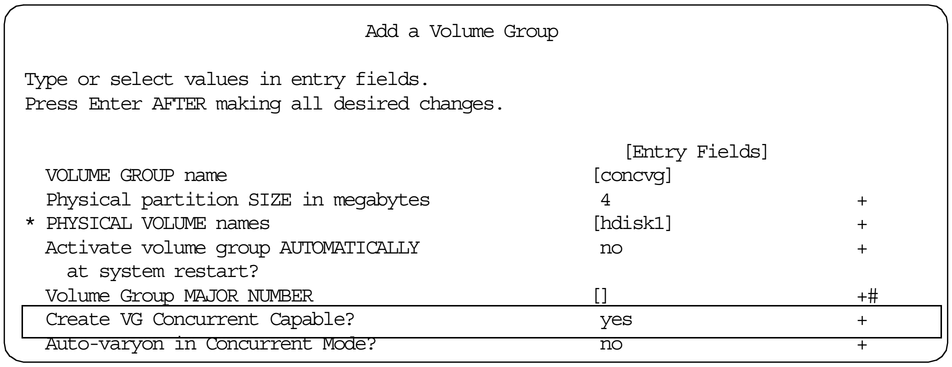
```
root@goofy:/ # smitty vg
    Add a Volume Group
```

In this panel, you have to specify the following necessary information:

- VOLUME GROUP name
- PHYSICAL VOLUME names
- Create VG Concurrent Capable

  You have to set this flag to yes (as shown in the rectangle in this screen).

Then press **Enter**.



```
                              Add a Volume Group

 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
   VOLUME GROUP name                                   [concvg]
   Physical partition SIZE in megabytes                 4                    +
 * PHYSICAL VOLUME names                               [hdisk1]              +
   Activate volume group AUTOMATICALLY                  no                   +
     at system restart?
   Volume Group MAJOR NUMBER                           []                    +#
   Create VG Concurrent Capable?                        yes                  +
   Auto-varyon in Concurrent Mode?                      no                   +
```

If you prefer the command line interface, you can use the following command line.

```
root@goofy:/ # mkvg -c -y concvg hdisk1
concvg
mkvg: This concurrent capable volume group must be varied on manually.
```

As stated by the result of the `mkvg` command, this volume group is not varied online upon the initial creation or system startup time.

4.2.2.2 Manual varyon with concurrent mode

To manually vary a concurrent access volume group online in concurrent mode using an AIX command, you can use the following SMIT panel.

```
root@goofy:/ # smitty vg
```

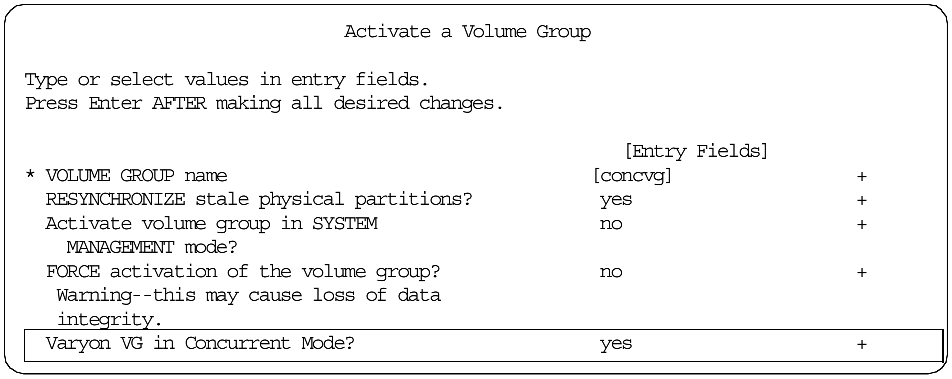
```
    Activate a Volume Group
```

In this panel, you have to specify the following mandatory information:

- VOLUME GROUP name
- Varyon VG in Concurrent Mode?

  You have to set this flag to yes (as shown in the rectangle in this screen).

Then press **Enter**.



```
                              Activate a Volume Group

 Type or select values in entry fields.
 Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
 * VOLUME GROUP name                                  [concvg]              +
   RESYNCHRONIZE stale physical partitions?            yes                  +
   Activate volume group in SYSTEM                     no                   +
     MANAGEMENT mode?
   FORCE activation of the volume group?               no                   +
    Warning--this may cause loss of data
    integrity.
   Varyon VG in Concurrent Mode?                       yes                  +
```

If you prefer the command line interface, you can use the following command line:

```
root@goofy:/ # varyonvg -c concvg
```

Please note that if you have a concurrent access volume group on non-SCSI RAID devices (like 7135, 7137, 2105 Versatile Storage Server), then you have to use the special command `convaryonvg` that is a part of the HACMP fileset cluster.base.server.events to bring it online manually with concurrent mode. To invoke this command, you can issue it like this:

```
# /usr/sbin/cluster/events/utils/convaryonvg concvg
```

For further information about this command, please refer following publication, *HACMP for AIX Administration Guide Version 4.3.1*, SC23-4279.

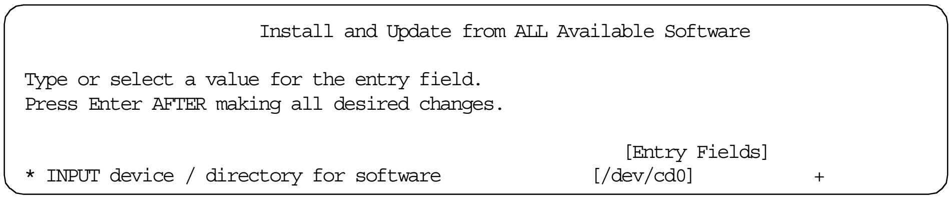
4.2.3 Installing the HACMP concurrent resource manager (CRM)

This section provides the results explained in Chapter 8 of the *HACMP Version 4.3 AIX: Installation Guide,* SC23-4278.

To install the concurrent access feature of HACMP, you must install the following fileset using SMIT:

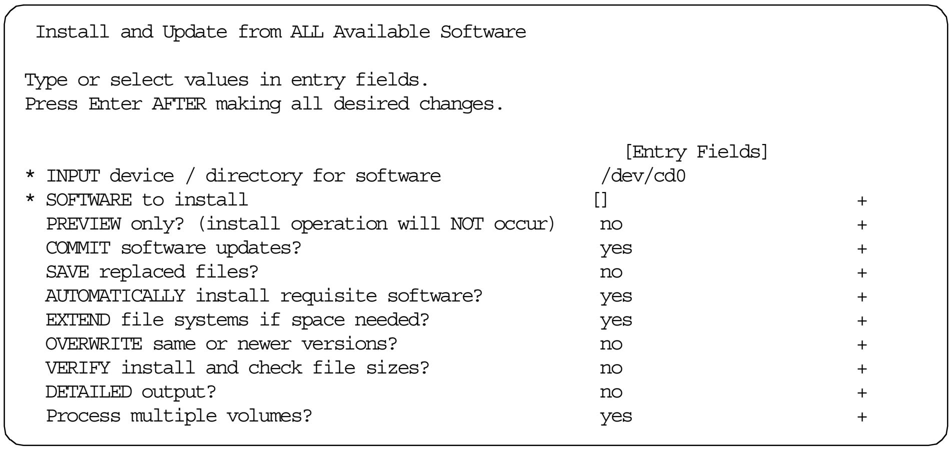
```
root@mickey:/ # smitty install_latest
```

Input the source device or directory name of the installation image, then press **Enter**.



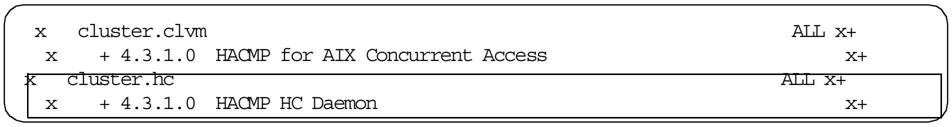
```
                     Install and Update from ALL Available Software

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* INPUT device / directory for software             [/dev/cd0]          +
```

Press the **F4** key on the SOFTWARE to install line:



```
 Install and Update from ALL Available Software

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* INPUT device / directory for software              /dev/cd0
* SOFTWARE to install                               []                    +
  PREVIEW only? (install operation will NOT occur)   no                   +
  COMMIT software updates?                           yes                  +
  SAVE replaced files?                               no                   +
  AUTOMATICALLY install requisite software?          yes                  +
  EXTEND file systems if space needed?               yes                  +
  OVERWRITE same or newer versions?                  no                   +
  VERIFY install and check file sizes?               no                   +
  DETAILED output?                                   no                   +
  Process multiple volumes?                          yes                  +
```

Choose the following package with the **F7** key, then press **Enter** twice.



```
 x   cluster.clvm                                                 ALL x+
  x    + 4.3.1.0  HACMP for AIX Concurrent Access                     x+
x   cluster.hc                                                   ALL x+
  x    + 4.3.1.0  HACMP HC Daemon                                     x+
```

Note that the fileset, cluser.hc.rte (shown as the rectangle in the above screen example), should be selected only if you have a configuration with a standalone RS/6000 cluster environment (not RS/6000 SP systems) and with Oracle Parallel Server running. For further detailed information about this fileset, please refer to 4.3.4, “What is the HC daemon?” on page 228.

After completion of this task, you will have the following filesets installed. The other filesets (except for the cluster.clvm.rte) are selected automatically.

```
root@mickey:/ # lslpp -L cluster.*
  Fileset                      Level  State  Description
  ----------------------------------------------------------------------------
  cluster.adt.client.demos   4.3.1.0    C    HACMP Client Demos
  cluster.adt.client.include
                             4.3.1.0    C    HACMP Client Include Files
  cluster.adt.client.samples.clinfo
                             4.3.1.0    C    HACMP Client CLINFO Samples
  cluster.adt.client.samples.clstat
                             4.3.1.0    C    HACMP Client Clstat Samples
  cluster.adt.client.samples.demos
                             4.3.1.0    C    HACMP Client Demos Samples
  cluster.adt.client.samples.libcl
                             4.3.1.0    C    HACMP Client LIBCL Samples
  cluster.adt.server.demos   4.3.1.0    C    HACMP Server Demos
  cluster.adt.server.samples.demos
                             4.3.1.0    C    HACMP Server Sample Demos
  cluster.adt.server.samples.images
                             4.3.1.0    C    HACMP Server Sample Images
  cluster.base.client.lib    4.3.1.0    C    HACMP Base Client Libraries
  cluster.base.client.rte    4.3.1.0    C    HACMP Base Client Runtime
  cluster.base.client.utils  4.3.1.0    C    HACMP Base Client Utilities
  cluster.base.server.diag   4.3.1.0    C    HACMP Base Server Diags
  cluster.base.server.events
                             4.3.1.0    C    HACMP Base Server Events
  cluster.base.server.rte    4.3.1.0    C    HACMP Base Server Runtime
  cluster.base.server.utils  4.3.1.0    C    HACMP Base Server Utilities
  cluster.clvm.rte           4.3.1.0    C    HACMP for AIX Concurrent Access
  cluster.cspoc.cmds         4.3.1.0    C    HACMP CSPOC Commands
  cluster.cspoc.dsh          4.3.1.0    C    HACMP CSPOC dsh and perl
  cluster.cspoc.rte          4.3.1.0    C    HACMP CSPOC Runtime Commands
  cluster.man.en_US.client.data
                             4.3.1.0    C    HACMP Client Man Pages - U.S.
                                             English
  cluster.man.en_US.cspoc.data
                             4.3.1.0    C    HACMP CSPOC Man Pages - U.S.
                                             English
  cluster.man.en_US.es.data  4.3.1.0    C    ES Man Pages - U.S. English
  cluster.man.en_US.server.data
                             4.3.1.0    C    HACMP Server Man Pages - U.S.
                                             English
  cluster.msg.En_US.client   4.3.1.0    C    HACMP Client Messages - U.S.
                                             English IBM-850
  cluster.msg.En_US.cspoc    4.3.1.0    C    HACMP CSPOC Messages - U.S.
                                             English IBM-850
  cluster.msg.En_US.server   4.3.1.0    C    HACMP Server Messages - U.S.
                                             English IBM-850
  cluster.msg.en_US.client   4.3.1.0    C    HACMP Client Messages - U.S.
                                             English
  cluster.msg.en_US.cspoc    4.3.1.0    C    HACMP CSPOC Messages - U.S.
                                             English
  cluster.msg.en_US.server   4.3.1.0    C    HACMP Server Messages - U.S.
                                             English
```

The concurrent resource manager fileset contains following files.

```
root@mickey:/ # lslpp -f cluster.clvm.rte
  Fileset               File
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  cluster.clvm.rte 4.3.1.0
                        /usr/sbin/cluster/clvm/.mode3_install
                        /usr/sbin/cluster/clvm/mode3_install
                        /usr/sbin/cluster
                        /usr/sbin/cluster/clvm

Path: /etc/objrepos
  cluster.clvm.rte 4.3.1.0
                        NONE
```

After the installation of the cluster.clvm fileset, the necessary modifications are made automatically by the post_i shell script (shown in the following example) called from the installp process. It releases the trigger that enables the concurrent mode varyon using the CLVM. In other words, without installing this fileset, CLVM does not have the capability to vary on the concurrent access volume group in concurrent mode.

```
root@mickey:/usr/sys/inst.images # strings cluster.clvm | sed '1,/post_i/d' | sed
'/^24/,$d'
#!/bin/ksh
# IBM_PROLOG_BEGIN_TAG
# This is an automatically generated prolog.
# Licensed Materials - Property of IBM
# (C) COPYRIGHT International Business Machines Corp. 1995,1998
# All Rights Reserved
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# IBM_PROLOG_END_TAG
# @(#)15  1.7  src/packages/cluster/clvm/rte/usr/cluster.clvm.rte.post_i.sh, pkg
hacmp, 43pkgha_rtr2, rtr2t3fa1 8/10/98 09:54:39
#   COMPONENT_NAME: pkghacmp
#   FUNCTIONS: if_failed
onintr
#   ORIGINS: 27
CLUSTDIR="/usr/sbin/cluster"
if_failed() {
        status=$1
        errmsg=$2
        if [ $status -ne 0 ]; then
                echo Failed $errmsg
                exit 99
        fi
onintr()
    echo Aborted by User.
trap onintr 1 2 3
###################### Begin CLVM license modifications #####################
/var/tmp/mkmode3_install > $CLUSTDIR/clvm/.mode3_install
if_failed $? "Configuring for Mode 3"
rm -f /var/tmp/mkmode3_install >/dev/null 2>&1
######################  End CLVM license modifications  #####################
exit 0
```

4.2.4 Installing HACMP cluster configurations: Standard method

This section provides a summary of the HACMP configuration steps described in the *HACMP Version 4.3 AIX: Installation Guide,* SC23-4278. The number represented in brackets matches the corresponding chapter number in the documentation. Some steps are covered in previous sections, and for others, we will only show the results instead of the full explanation of the installation step, since the HACMP configuration steps are not the primary goal of this book (these steps are marked as Skipped or Omitted). Only the steps related to concurrent access volume groups are covered.

1. Configuring networks [4]

   Covered in 4.2.1, “Example configuration” on page 195.

2. Installing Shared Disk Devices [5]

   Covered in 4.2.1, “Example configuration” on page 195.

3. Defining Shared LVM Components [6]

   Covered in 4.2.2, “Defining shared LVM components for concurrent access” on page 198.

4. Additional AIX administrative tasks [7]

   Omitted.

5. Installation HACMP for AIX software [8]

   Covered in 4.2.3, “Installing the HACMP concurrent resource manager (CRM)” on page 200.

6. Upgrading an HACMP Cluster [9]

   Omitted, since this installation is a brand new installation.

7. Verifying cluster software [10]

   Omitted.

8. Defining the cluster topology [11]

   Done as follows (actually, this step is done using the shell script shown in Appendix B.1, “Define-Cluster-Topology” on page 357):

   a. Defining the cluster

   ```
   root@mickey:/ # cllsclstr
   ID      Name    Security

   1       disney  Standard
   ```

   b. Defining nodes

   ```
   root@mickey:/ # odmget HACMPnode | grep name | sort | uniq
           name = "goofy"
           name = "mickey"
   ```

   c. Defining adapters

   ```
   root@mickey:/ # cllsif
   Adapter              Type       Network    Net Type   Attribute  Node       IP
   Address       Hardware Address

   goofy_en0            service    ETHER_ITSC ether      public     goofy
   100.100.100.2
   goofy_tty            service    RS232_ITSC rs232      serial     goofy
   /dev/tty0
   ```

```
goofy_boot           boot       TRN_ITSC   token      public     goofy
9.3.187.186
goofy                service    TRN_ITSC   token      public     goofy
9.3.187.185
goofy_stby           standby    TRN_ITSC   token      public     goofy
9.3.187.252
mickey_en0           service    ETHER_ITSC ether      public     mickey
100.100.100.1
mickey_tty           service    RS232_ITSC rs232      serial     mickey
/dev/tty0
mickey_boot          boot       TRN_ITSC   token      public     mickey
9.3.187.184
mickey               service    TRN_ITSC   token      public     mickey
9.3.187.183
mickey_stby          standby    TRN_ITSC   token      public     mickey
9.3.187.251
```

d. Configuring network modules

   Omitted.

e. Synchronizing the cluster definition across nodes

```
mickey:/ # clconfig -s -t
```

As a result of this step, we have the following HACMP topology definitions:

```
root@mickey:/ # cllsnode

NODE goofy:
        Interfaces to network ETHER_ITSC
                service Interface: Name goofy_en0, Attribute public, IP address
100.100.100.2
        Interfaces to network RS232_ITSC
                service Interface: Name goofy_tty, Attribute serial, IP address
/dev/tty0
        Interfaces to network TRN_ITSC
                boot Interface: Name goofy_boot, Attribute public, IP address
9.3.187.186
               service Interface: Name goofy, Attribute public, IP address 9.3.187.185
                standby Interface: Name goofy_stby, Attribute public, IP address
9.3.187.252

NODE mickey:
        Interfaces to network ETHER_ITSC
                service Interface: Name mickey_en0, Attribute public, IP address
100.100.100.1
        Interfaces to network RS232_ITSC
                service Interface: Name mickey_tty, Attribute serial, IP address
/dev/tty0
        Interfaces to network TRN_ITSC
                boot Interface: Name mickey_boot, Attribute public, IP address
9.3.187.184
                service Interface: Name mickey, Attribute public, IP address
9.3.187.183
                standby Interface: Name mickey_stby, Attribute public, IP address
9.3.187.251
```

9. Configuring Cluster Resources [12]

   Done as follows (actually, this step is done using the script shown in Appendix B.2, “Configuring-Cluster-Resources” on page 360):

a. Configuring application servers

Omitted.

b. Configuring AIX Fast Connect resources

Skipped. This cluster does not have AIX Fast Connect application installed.

c. Configuring AIX Connections services

Skipped. This cluster does not have AIX Connections application installed.

d. Configuring CS/AIX Communications links

Skipped. This cluster does not have CS/AIX application installed.

e. Creating Resources Groups

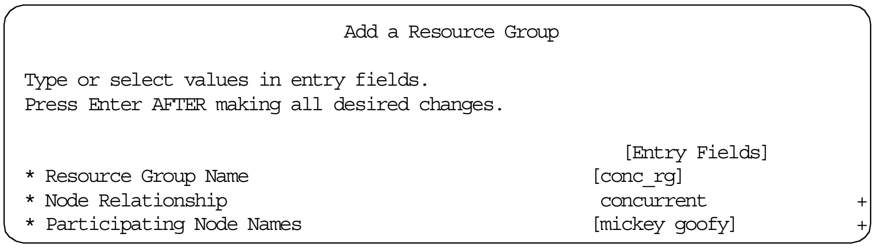
To create the concurrent resource group, you can use the following SMIT panel:

```
root@mickey:/ # smitty hacmp
    Cluster Configuration
        Cluster Resources
            Define Resource Groups
                Add a Resource Group
```

Specify the necessary information. In this example, we input the following values:

- conc_rg as Resource Group Name
- concurrent as Node Relationship
- mickey and goofy as Participating Node Names

Then press **Enter**.



```
                              Add a Resource Group

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                        [Entry Fields]
* Resource Group Name                                [conc_rg]
* Node Relationship                                   concurrent              +
* Participating Node Names                           [mickey goofy]           +
```

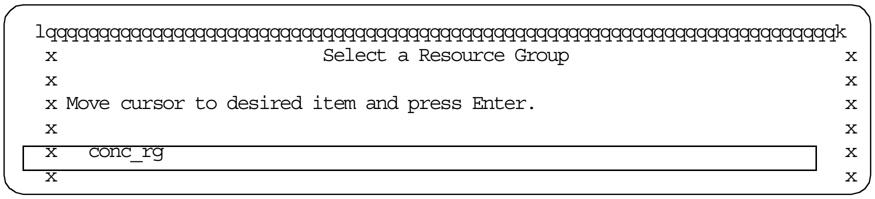
f. Configuring Resources for Resource Groups

To add resources to a resource group, you can use the following SMIT panel:

```
root@mickey:/ # smitty hacmp
        Cluster Configuration
                Cluster Resources
```

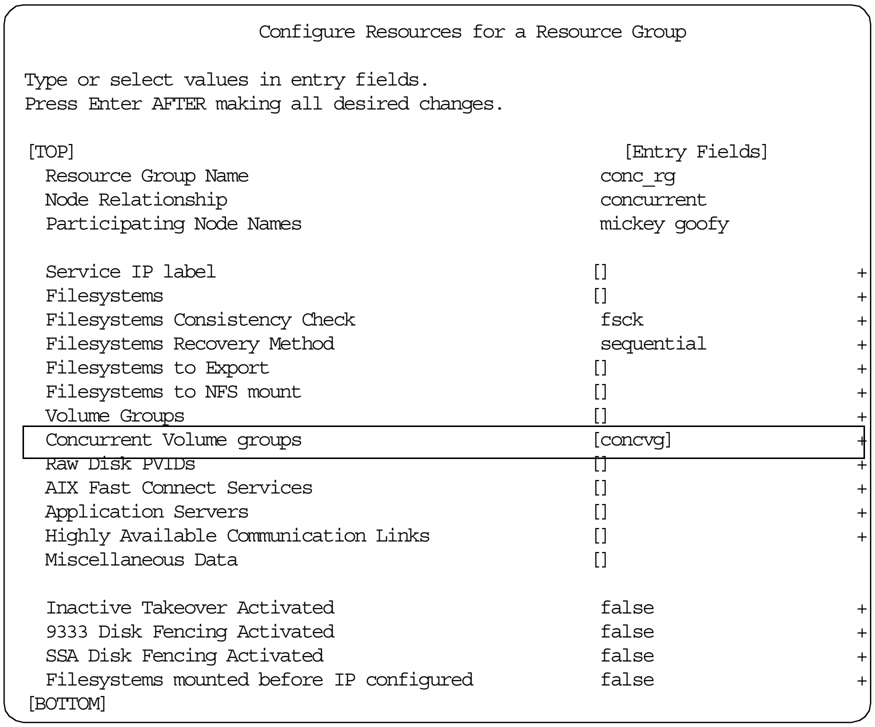
```
                    Change/Show Resources for a Resource Group
```

Choose the conc_rg as Resource Group, then press **Enter**.



```
lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
x                         Select a Resource Group                             x
x                                                                             x
x Move cursor to desired item and press Enter.                                x
x                                                                             x
x   conc_rg                                                                   x
x                                                                             x
```

Specify the necessary information. In this example, input concvg as Concurrent Volume Groups (shown as the rectangle). Then press **Enter**.



```
                     Configure Resources for a Resource Group

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                  [Entry Fields]
  Resource Group Name                                  conc_rg
  Node Relationship                                    concurrent
  Participating Node Names                             mickey goofy

  Service IP label                                    []                    +
  Filesystems                                         []                    +
  Filesystems Consistency Check                        fsck                 +
  Filesystems Recovery Method                          sequential           +
  Filesystems to Export                               []                    +
  Filesystems to NFS mount                            []                    +
  Volume Groups                                       []                    +
  Concurrent Volume groups                            [concvg]              +
  Raw Disk PVIDs                                      []                    +
  AIX Fast Connect Services                           []                    +
  Application Servers                                 []                    +
  Highly Available Communication Links                []                    +
  Miscellaneous Data                                  []

  Inactive Takeover Activated                          false                +
  9333 Disk Fencing Activated                          false                +
  SSA Disk Fencing Activated                           false                +
  Filesystems mounted before IP configured             false                +
[BOTTOM]
```

Note

You should not input anything except for the concurrent access volume group names in the concurrent resource group. The service address and file system resources can not be the resource of the concurrent resource group.

g. Synchronizing cluster resources

```
# clconfig -s -r
```

h. Configuring run-time parameters

   Omitted.

As a result of this step, we have the following HACMP concurrent resource group definitions.

```
root@mickey:/ # cllsres -g conc_rg
CONCURRENT_VOLUME_GROUP="concvg"
DISK_FENCING="false"
FSCHECK_TOOL="fsck"
FS_BEFORE_IPADDR="false"
INACTIVE_TAKEOVER="false"
RECOVERY_METHOD="sequential"
SSA_DISK_FENCING="false"
```

In addition to this concurrent resource group, the following two cascading resource groups are also defined:

```
root@mickey:/ # cllsgrp
1_2_rg
2_1_rg
conc_rg
root@mickey:/ # cllsres -g 1_2_rg
SERVICE_LABEL="mickey"
root@mickey:/ # cllsres -g 2_1_rg
SERVICE_LABEL="goofy"
```

These resource group definitions are also shown in graphical format in Figure 70.

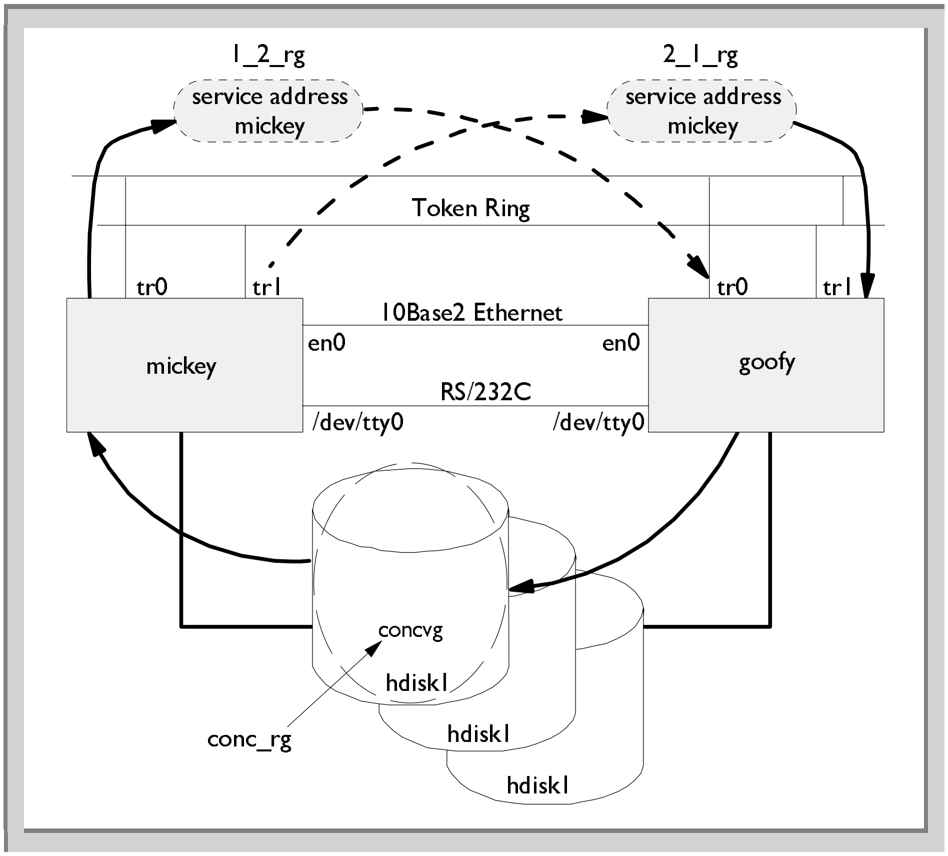


Figure 70. Defined resource groups

10.Customizing cluster events and log files [13]

Omitted.

11.Setting up clinfo on server nodes [14]

Omitted.

12.Supporting AIX error notification [15]

Omitted.

4.2.5 Managing shared logical volumes for concurrent access

The managing tasks of shared logical volumes for concurrent access are categorized in the following list.

- Creating a logical volume in a concurrent access volume group

- Deleting a logical volume in a concurrent access volume group
- Changing some attributes of a logical volume in a concurrent access volume group
- Adding or removing copies of mirrored logical volume in concurrent access volume group

To perform the above tasks, you have the three following choices:

- Static ODM update method (apply to SCSI RAID system management).

  This method creates the logical volumes on one system that vary online the concurrent access volume group with *non-concurrent* mode, then propagates this configuration to the other nodes statically.

  Every release of AIX and HACMP/CRM support this traditional method. This was the only supported way before AIX Version 4.3.0 and HACMP/CRM Version 4.3.0.

- Dynamic ODM update method.

  This method creates the logical volumes on one system (initiator node) that vary online the concurrent access volume group with *concurrent* mode, then immediately propagates this configuration to the other nodes dynamically.

  You need at least AIX Version 4.3.0 and HACMP Version 4.3.0 to use this method. It exploits the enhancement brought to the AIX LVM for the concurrent access discussed in 4.1.3.2, “Vary on with concurrent mode without prerequisites” on page 190.

These two methods are described in the following sections.

4.2.6 Static ODM update method

As described before, this traditional method was the only officially supported one prior to AIX Version 4.3.0 and the HACMP/CRM Version 4.3.0. In earlier versions of AIX, the LVM does not provide the function that enables dynamically creating, removing, and changing the logical volumes on the concurrent access volume group with concurrent mode (it also includes the dynamic updates of the LVM definition in the ODM among the cluster nodes). This method is also valid for SCSI RAID systems.

It requires the shutdown of the concurrent access volume group with concurrent mode, but it is a safe method and remains valid even with the latest versions of the AIX and HACMP/CRM software.

The brief work flow is presented in the following list, but the basic activity of this method is the same as the method managing the shared LVM components on non-concurrent access volume group. Therefore detailed explanation about this method is omitted here. For complete information about this method, please consult the *HACMP for AIX Version 4.3.1 Installation Guide.*

1. Vary offline the volume group on all the cluster nodes. This step is also achieved by stopping the cluster manager on all the cluster nodes.
2. Export the volume group from all the cluster nodes except for one node (called the source node).
3. On the source node:
   a. Vary online the volume group in non-concurrent mode.
   b. Create, remove, and change the logical volumes in the volume group.
   c. Change the autovaryon attribute set to no on this volume group.
   d. Vary offline the volume group.
4. On the other nodes, process the following steps:
   a. Import the volume group.
   b. Examine the necessary LVM definition in the ODM.
   c. Change the autovaryon attribute set to no on this volume group.
   d. Vary offline the volume group.

4.2.7 Dynamic ODM update method

As described before, from AIX Version 4.3.0, the AIX LVM provides the necessary function for this method. If you have installed the HACMP/CRM, there are two ways for this method. One uses the standard AIX LVM function, and the other is using the HACMP C-SPOC (Cluster-Single Point Of Control) function.

We are providing the brief examples that treat creating and removing the shared logical volume using those two methods.

Note that the C-SPOC is an administrative utility provided by the HACMP product to enable system administration tasks consistently on all the cluster nodes. Basically, it consists of some scripts and rsh facility. For further detailed information about the C-SPOC, refer to *HACMP for AIX Version 4.3.1 Concepts and Facilities*, SC23-4276.

4.2.7.1 Managing the shared logical volume using standard function

In the following example, all the cluster nodes have to vary the concurrent access volume group online with concurrent mode. Otherwise, the necessary ODM updates are not performed on that node.

Creating the shared logical volume using standard function

To create the shared logical volume with concurrent mode using the standard AIX LVM function, you can use the following SMIT interface on one of the cluster nodes (called the initiator node).

```
root@mickey:/ # smitty lvm
    Logical Volumes
        Add a Logical Volume
```

Choose the concvg as Volume Group name, then press **Enter**.

```
lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
x                       Concurrent Volume Group Names                        x
x                                                                            x
x Move cursor to desired item and press Enter.                               x
x                                                                            x
x   conc_rg concvg                                                           x
x                                                                            x
```

Specify the following mandatory information:

- Number of LOGICAL PARTITIONS
- Logical Volume NAME

Then press **Enter**.

Note

You should change the following attribute to 'no' in the following example:

- Mirror Write Consistency
- Enable BAD Block relocation

These are shown in the rectangle in the following display.

```
                                Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                    [Entry Fields]
  Logical volume NAME                                 [conclv]
* VOLUME GROUP name                                    concvg
* Number of LOGICAL PARTITIONS                        [20]                    #
  PHYSICAL VOLUME names                               []                     +
  Logical volume TYPE                                 []
  POSITION on physical volume                          middle                +
  RANGE of physical volumes                            minimum               +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  []                      #
    to use for allocation
  Number of COPIES of each logical                     1                     +
    partition
  Mirror Write Consistency?                            no                    +
  Allocate each logical partition copy                 yes                   +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                   yes                   +
    reorganization?
  Logical volume LABEL                                []
  MAXIMUM NUMBER of LOGICAL PARTITIONS                [512]                   #
  Enable BAD BLOCK relocation?                         no                    +
  SCHEDULING POLICY for reading/writing                parallel              +
    logical partition copies
  Enable WRITE VERIFY?                                 no                    +
  File containing ALLOCATION MAP                      []
  Stripe Size?                                        [Not Striped]          +
[BOTTOM]
```

Then you will see the following processes (shown in the rectangles) running on the other cluster node(s).

```
root@goofy:/ # ps -ef
(many lines are snipped off)
    root  8778  3646   0   Sep 22      -  0:00 /usr/sbin/rpc.mountd
    root  9050     1   0   Sep 22      -  0:03 /usr/sbin/cron
    root  9300  3646   0   Sep 22      -  0:01 /usr/sbin/rpc.lockd
    root  9574     1   0   Sep 22      -  0:00 /usr/lpp/diagnostics/bin/diagd
    root  9868 13530   0 17:07:19      -  0:00 ksh /usr/sbin/synclvodm -c -F -
    root 10326  3646   0   Sep 22      -  0:00 /usr/sbin/qdaemon
    root 10594  3646   0   Sep 22      -  0:00 /usr/sbin/writesrv
    root 11096     1   0   Sep 23   lft0  0:00 /usr/sbin/getty /dev/console
    root 11352     1   0   Sep 22      -  0:00 /usr/lpp/csd/bin/hc
    root 12000  7118  24 17:07:26  pts/0  0:00 ps -ef
    root 12310  9868  21 17:07:20      -  0:00 ksh /usr/sbin/updatelv -c conc1
    root 13530  3646   0 22:01:42      -  0:00 /usr/sbin/clvmd
    root 15414 12310   3 17:07:25      -  0:00 getlvcb -aceLrsSPtu conc1lv
```

After completion of this task, you can see that the logical volume is defined on both nodes. It is a proof that the necessary LVM definition in the ODM is dynamically updated on the passive node (node goofy).

- On mickey

```
root@mickey:/ # lslv conclv
LOGICAL VOLUME:     conclv                 VOLUME GROUP:   concvg
LV IDENTIFIER:      000010736127caf7.3     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        4 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                20                     PPs:            20
STALE PPs:          0                      BB POLICY:      non-relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV?: yes
```

- On goofy

```
root@goofy:/ # lslv conclv
LOGICAL VOLUME:     conclv                 VOLUME GROUP:   concvg
LV IDENTIFIER:      000010736127caf7.3     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        4 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                20                     PPs:            20
STALE PPs:          0                      BB POLICY:      non-relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV?: yes
```

Note that you should not confuse this behavior with the *lazy update* function provided by HACMP since HACMP Version 4.2.2.

Accessing the shared logical volume from both nodes

You can confirm the accessibility of this logical volume using the `dd` command from both nodes.

Note

Do not overwrite the LVCB of the logical volume (you have to skip the first disk block of the logical partition if accessing it as a raw device).

- Write from goofy

```
root@goofy:/ # dd if=/etc/hosts of=/dev/conc1lv seek=1
1+1 records in.
1+1 records out.
```

- Read from mickey

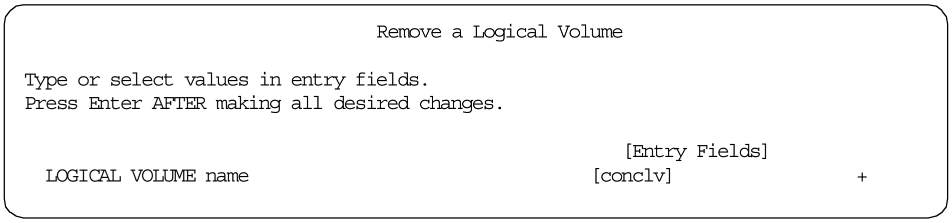
```
root@mickey:/ # dd if=/dev/conc1lv skip=1 count=1
# /etc/hosts -- local host table.
127.0.0.1             loopback localhost      # loopback (lo0) name/address
########################################################################
#   Lab Cluster
########################################################################
#
#   mickey
#
#100.100.101.21        mickey_atboot
100.100.100.1          mickey_en0 mickey
100.100.101.1          mickey_at0
100.100.102.1          mickey_at1
9.3.187.183            mickey
9.3.187.184            mickey_boot
9.3.187.251            mickey_stby
#
#   goofy
#
#100.100.101.22        goofy_atboot
100.100.1+0 records in.
1+0 records out.
```

Removing the shared logical volume using standard function

To remove the shared logical volume with concurrent mode using the standard AIX LVM function, you can use the following SMIT interface on one of the cluster nodes (called initiator node).

```
root@mickey:/ # smitty lvm
    Logical Volumes
        Remove a Logical Volume
```

Choose the conclv as LOGICAL VOLUME name, then press **Enter**.



```
                              Remove a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
  LOGICAL VOLUME name                             [conclv]              +
```

After completion of this task, you can see that the logical volume has been removed on both nodes.

4.2.7.2 Managing the shared logical volume using C-SPOC

Strictly speaking, there is no distinct functional difference with using the standard AIX LVM function or the C-SPOC function to create or remove the shared logical volumes, since the C-SPOC just utilizes the standard AIX LVM function for this purpose (though it has a capability to invoke another process on all the cluster nodes concurrently). But, remember that the C-SPOC gives you the following various administrative functions that can be performed on all the cluster nodes in a consistent manner:

- Maintain user and group accounts
- Managing the shared LVM components
- Control HACMP services on a cluster-wide basis

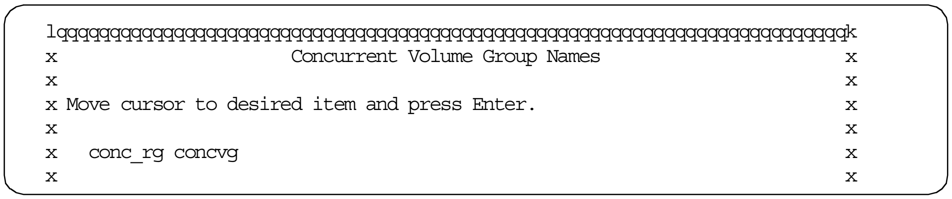
So, using the C-SPOC is a good discipline for managing the cluster. Therefore, we include the following example that produces the same result as the example shown in the previous section.

Creating the shared logical volume using C-SPOC

To create the shared logical volume with concurrent mode and C-SPOC, you can use the following SMIT interface on one of the cluster nodes (called the initiator node).

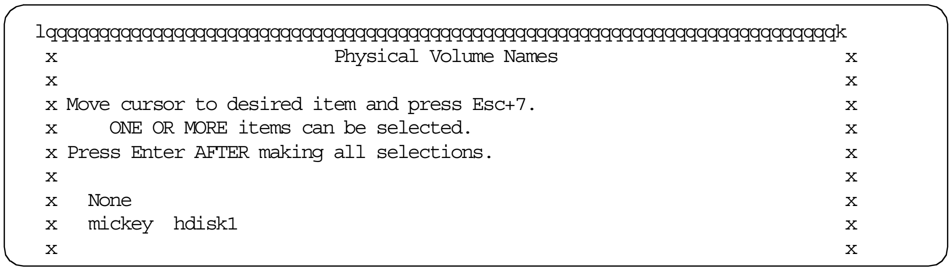
```
root@mickey:/ # smitty hacmp
    Cluster System Management
        Cluster Concurrent Logical Volume Manager
            Concurrent Logical Volumes
                Add a Concurrent Logical Volume
```

Choose the concvg as Concurrent Volume Group Names, then press **Enter**.



```
lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
x                       Concurrent Volume Group Names                     x
x                                                                         x
x Move cursor to desired item and press Enter.                            x
x                                                                         x
x   conc_rg concvg                                                        x
x                                                                         x
```

Choose the hdisk1 as Physical Volume Names, then press enter.



```
lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
 x                           Physical Volume Names                        x
 x                                                                        x
 x Move cursor to desired item and press Esc+7.                           x
 x     ONE OR MORE items can be selected.                                 x
 x Press Enter AFTER making all selections.                               x
 x                                                                        x
 x   None                                                                 x
 x   mickey  hdisk1                                                       x
 x                                                                        x
```

Specify the following necessary information.

- Number of LOGICAL PARTITIONS

- Logical Volume NAME

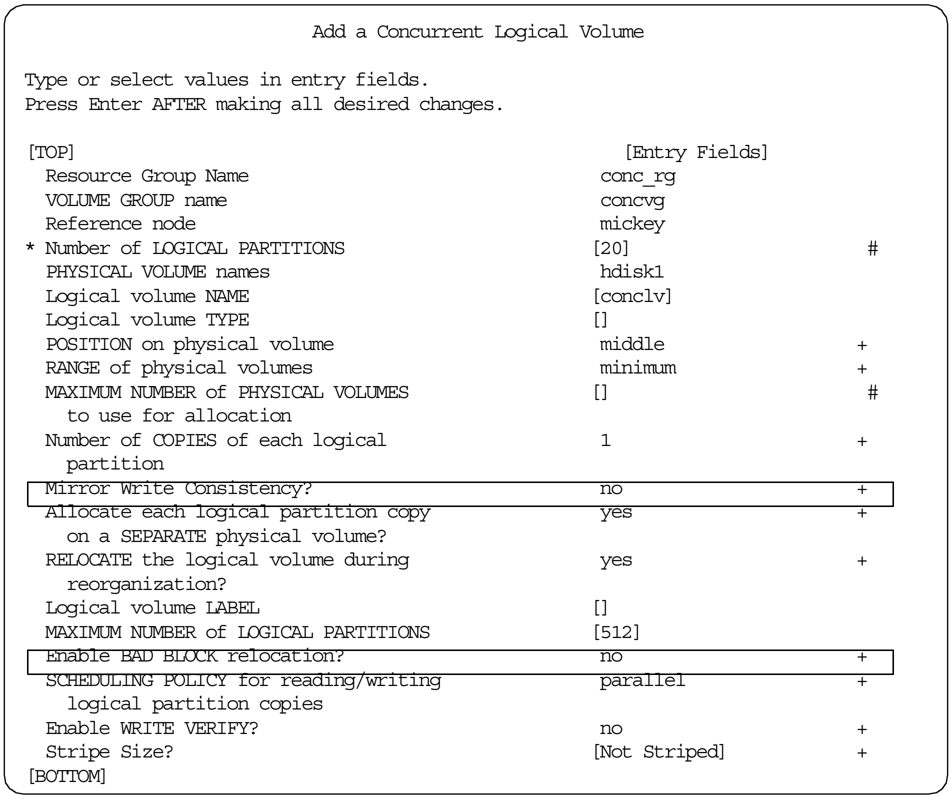
Then press **Enter**.

Note

You should change the following attribute to ‘no’ in the following example.

- Mirror Write Consistency?
- Enable BAD Block relocation?

These are shown in the rectangle in the following display.



```
                              Add a Concurrent Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                    [Entry Fields]
  Resource Group Name                                  conc_rg
  VOLUME GROUP name                                    concvg
  Reference node                                       mickey
* Number of LOGICAL PARTITIONS                        [20]                     #
  PHYSICAL VOLUME names                                hdisk1
  Logical volume NAME                                 [conclv]
  Logical volume TYPE                                 []
  POSITION on physical volume                          middle                 +
  RANGE of physical volumes                            minimum                +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  []                       #
    to use for allocation
  Number of COPIES of each logical                     1                      +
    partition
  Mirror Write Consistency?                            no                     +
  Allocate each logical partition copy                 yes                    +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                   yes                    +
    reorganization?
  Logical volume LABEL                                []
  MAXIMUM NUMBER of LOGICAL PARTITIONS                [512]
  Enable BAD BLOCK relocation?                         no                     +
  SCHEDULING POLICY for reading/writing                parallel               +
    logical partition copies
  Enable WRITE VERIFY?                                 no                     +
  Stripe Size?                                        [Not Striped]           +
[BOTTOM]
```

After completion of this task, you can see the logical volume is defined on both nodes.

- On mickey

```
root@mickey:/ # lslv conclv
LOGICAL VOLUME:     conclv                VOLUME GROUP:   concvg
```

```
LV IDENTIFIER:      000010736127caf7.3     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        4 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                20                     PPs:            20
STALE PPs:          0                      BB POLICY:      non-relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV?: yes
```

- On goofy

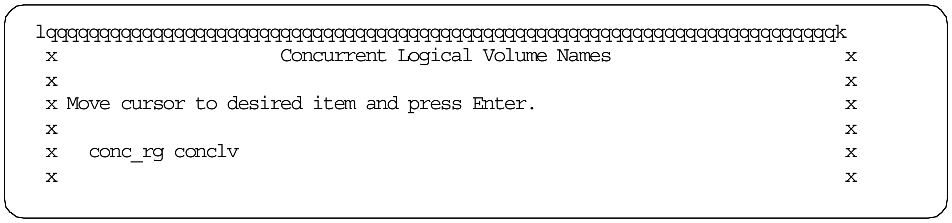
```
root@goofy:/ # lslv conclv
LOGICAL VOLUME:     conclv                 VOLUME GROUP:   concvg
LV IDENTIFIER:      000010736127caf7.3     PERMISSION:     read/write
VG STATE:           active/complete        LV STATE:       closed/syncd
TYPE:               jfs                    WRITE VERIFY:   off
MAX LPs:            512                    PP SIZE:        4 megabyte(s)
COPIES:             1                      SCHED POLICY:   parallel
LPs:                20                     PPs:            20
STALE PPs:          0                      BB POLICY:      non-relocatable
INTER-POLICY:       minimum                RELOCATABLE:    yes
INTRA-POLICY:       middle                 UPPER BOUND:    32
MOUNT POINT:        N/A                    LABEL:          None
MIRROR WRITE CONSISTENCY: off
EACH LP COPY ON A SEPARATE PV?: yes
```

Removing the shared logical volume using C-SPOC

To remove the shared logical volume with concurrent mode and C-SPOC, you can use the following SMIT interface on one of the cluster nodes (called initiator node).

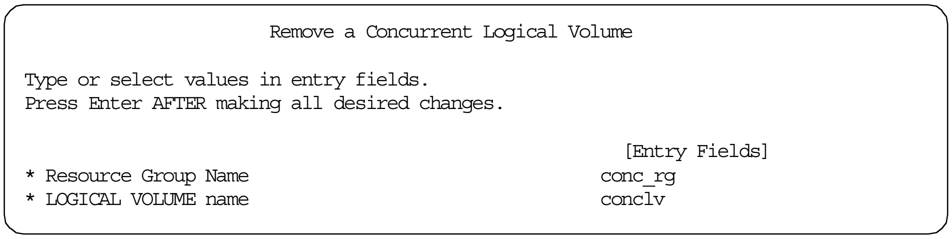
```
root@mickey:/ # smitty hacmp
    Cluster System Management
        Cluster Concurrent Logical Volume Manager
            Concurrent Logical Volumes
                Remove a Concurrent Logical Volume
```

Choose the conclv as Concurrent Logical Volume Names, then press **Enter**.



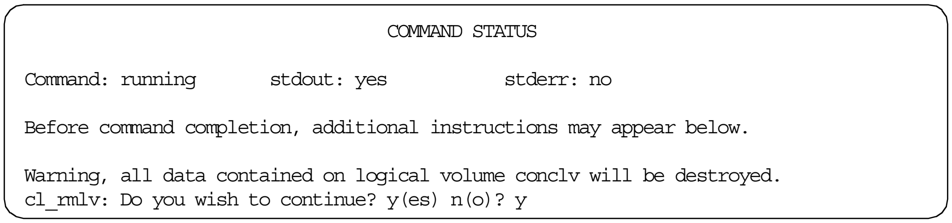
```
lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
 x                     Concurrent Logical Volume Names                      x
 x                                                                          x
 x Move cursor to desired item and press Enter.                             x
 x                                                                          x
 x   conc_rg conclv                                                         x
 x                                                                          x
```

Confirm the logical volume name to be removed, then press **Enter**.



```
                       Remove a Concurrent Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
* Resource Group Name                                   conc_rg
* LOGICAL VOLUME name                                   conclv
```

It requests your confirmation, then press **y** and **Enter**.



```
                                  COMMAND STATUS

Command: running      stdout: yes           stderr: no

Before command completion, additional instructions may appear below.

Warning, all data contained on logical volume conclv will be destroyed.
cl_rmlv: Do you wish to continue? y(es) n(o)? y
```

After completion of this task, you can see the logical volume has been removed on both nodes.

4.3 Shared disk environment solutions

This section provides a brief summary for the shared disk environment solutions available on RS/6000 platforms. Roughly categorized, there are two platform types.

- RS/6000 cluster systems
- RS/6000 Scalable Parallel (SP) systems

On RS/6000 SP systems, in addition to the concurrent access, you can select another shared disk environment solution as discussed in 4.3.3, “Shared disk solution on an RS/6000 SP system” on page 226.

4.3.1 HACMP and HACMP Enhanced Scalability (ES)

As stated before, the Concurrent Resource Manager (CRM) feature of the HACMP product is a mandatory component to enable concurrent access. Besides the CRM feature, current HACMP products have two different implementation branches. The first one is the original HACMP (also called, HACMP classic). The other is HACMP Enhanced Scalability (abbreviated HACMP/ES, also known as HAES). Why was HACMP/ES developed when there was already an HACMP? The answer scalability. There was a huge demand to extend the maximum number of nodes in one cluster. To address this functional enhancement, the HACMP/ES has a different communication method among the cluster nodes compare with HACMP classic, called NIM (Network Interface Module). In the HACMP/ES, it is called Topology Service/ES or RSCT (Risc System Cluster Technology), depending on the version. To use this communication method, the current HACMP/ES extends the maximum cluster node number from the eight of the HACMP to 32.

The following tables are provided to show the functional difference about the concurrent access, based on the software, HACMP or HACMP/ES, and the platforms, RS/6000 cluster or RS/6000 SP, for the HACMP Version 4.1.0 and later.

Table 15. HACMP Version 4.1.0, 4.1.1 (Program Number 5696-933)

HACMP product		**RS/6000 cluster systems**		**RS/6000 SP systems**	
HACMP	VG mode	normal	concurrent	normal	concurrent
	Supported	Yes	Yes[a]	Yes	Yes[a]

[a] Requires the CRM feature of HACMP

- There was no HACMP/ES in HACMP Version 4.1.0, 4.1.1.
- Supported on AIX Version 4.1.

Table 16. HACMP Version 4.2.0, 4.2.1, 4.2.2 (Program Number 5765-A86)

HACMP product		**RS/6000 cluster systems**		**RS/6000 SP systems**	
HACMP	VG mode	normal	concurrent	normal	concurrent
	Supported	Yes	Yes[a]	Yes	Yes[a]
HACMP/ ES	VG mode	normal	concurrent	normal	concurrent
	Supported	No	No	Yes	No

[a] Requires the CRM feature of HACMP

- There was no HACMP/ES derivative in HACMP Version 4.2.0.
- Supported on AIX Version 4.2.

- This version HACMP/ES depends on Topology Service/ES function provided by PSSP (Parallel System Support Program) Version 2.3 and 2.4. Therefore, there is no support on RS/6000 cluster platforms.

Table 17. HACMP Version 4.3.0, 4.3.1 (Program Number 5765-D28)

HACMP product		**RS/6000 cluster systems**		**RS/6000 SP systems**	
HAS	VG mode	normal	concurrent	normal	concurrent
	Supported	Yes	Yes[a]	Yes	Yes[a]
HAES	VG mode	normal	concurrent	normal	concurrent
	Supported	Yes	Yes[b]	Yes	Yes[b]

[a] Requires the CRM feature of HACMP

[b] Requires the CRM feature of HACMP/ES

- Supported on AIX Version 4.3.
- This version of HACMP/ES depends on the RSCT function that is a separate package from the PSSP Version 3.1. (The HACMP/ES also includes the RSCT. Therefore, HACMP/ES does not have prerequisite with PSSP.)

We chose the HACMP/CRM to avoid complexity in this chapter example, but, if you want to use the HACMP/ES/CRM, you have to install the following filesets:

- Select the rsct.* filesets that are components of the RSCT function.
- Select the cluster.es.clvm.rte instead of the cluster.clvm.rte fileset.
- Select the cluster.es.hc.rte instead of cluster.hc.rte fileset (if applicable).

```
Fileset Name                Level                     I/U Q Content
  ====================================================================
cluster.adt.es.client.demos 4.3.1.0                    I  N usr
#   ES Client Demos

  cluster.adt.es.client.include 4.3.1.0                I  N usr
#   ES Client Include Files

  cluster.adt.es.client.samples.clinfo 4.3.1.0         I  N usr
#   ES Client CLINFO Samples

  cluster.adt.es.client.samples.clstat 4.3.1.0         I  N usr
#   ES Client Clstat Samples

  cluster.adt.es.client.samples.demos 4.3.1.0          I  N usr
#   ES Client Demos Samples

  cluster.adt.es.client.samples.libcl 4.3.1.0          I  N usr
#   ES Client LIBCL Samples
```

```
  cluster.adt.es.java.demo.monitor 4.3.1.0             I  N usr
#   ES Web Based Monitor Demo

  cluster.adt.es.server.demos 4.3.1.0                  I  N usr
#   ES Server Demos

  cluster.adt.es.server.samples.demos 4.3.1.0          I  N usr
#   ES Server Sample Demos

cluster.adt.es.server.samples.images 4.3.1.0           I  N usr
#   ES Server Sample Images

cluster.es.client.lib         4.3.1.0                  I  N usr
#   ES Client Libraries

  cluster.es.client.rte       4.3.1.0                  I  N usr,root
#   ES Client Runtime

  cluster.es.client.utils     4.3.1.0                  I  N usr
#   ES Client Utilities

  cluster.es.server.diag      4.3.1.0                  I  N usr
#   ES Server Diags

  cluster.es.server.events    4.3.1.0                  I  N usr
#   ES Server Events

  cluster.es.server.rte       4.3.1.0                  I  N usr,root
#   ES Base Server Runtime

  cluster.es.server.utils     4.3.1.0                  I  N usr,root
#   ES Server Utilities

  cluster.es.clvm.rte         4.3.1.0                  I  N usr,root
#   ES for AIX Concurrent Access

  cluster.es.cspoc.cmds       4.3.1.0                  I  N usr
#   ES CSPOC Commands

  cluster.es.cspoc.dsh        4.3.1.0                  I  N usr
#   ES CSPOC dsh and perl

  cluster.es.cspoc.rte        4.3.1.0                  I  N usr
#   ES CSPOC Runtime Commands

  cluster.es.hc.rte           4.3.1.0                  I  N usr,root
#   ES HC Daemon

  cluster.es.taskguides.shrvolgrp 4.3.1.0              I  N usr
#   ES Shr Vol Grp Task Guides

cluster.man.en_US.client.data 4.3.1.0                  I  N share
#   HACMP Client Man Pages - U.S. English

  cluster.man.en_US.cspoc.data 4.3.1.0                 I  N share
#   HACMP CSPOC Man Pages - U.S. English

  cluster.man.en_US.server.data 4.3.1.0                I  N share
#   HACMP Server Man Pages - U.S. English

  cluster.man.en_US.es.data   4.3.1.0                  I  N share
#   ES Man Pages - U.S. English
```

```
  cluster.msg.en_US.es.client 4.3.1.0                  I  N usr
#   ES Client Messages - U.S. English

  cluster.msg.en_US.es.server 4.3.1.0                  I  N usr
#   ES Recovery Driver Messages - U.S. English

  cluster.vsm.es              4.3.1.0                  I  N usr
#   ES VSM Configuration Utility

  rsct.basic.hacmp            1.1.1.0                  I  N usr
#   RS/6000 Cluster Technology basic function (HACMP domain)

  rsct.basic.rte              1.1.1.0                  I  N usr,root
#   RS/6000 Cluster Technology basic function (all domain)

  rsct.basic.sp               1.1.1.0                  I  N usr
#   RS/6000 Cluster Technology basic function (SP domain)

  rsct.clients.hacmp          1.1.1.0                  I  N usr
#   RS/6000 Cluster Technology client function (HACMP domain)

  rsct.clients.rte            1.1.1.0                  I  N usr
#   RS/6000 Cluster Technology client function (all domain)

  rsct.clients.sp             1.1.1.0                  I  N usr
#   RS/6000 Cluster Technology client function (SP domain)
```

For further detailed information about the HACMP product and its features, please consult with appropriate announcement letters.

4.3.2 Cluster lock manager

The cluster lock manager is a daemon subsystem that provides advisory locking services that allow concurrent applications running on multiple nodes in an HACMP cluster to coordinate their use of shared resources (mainly the shared logical volumes accessed using raw I/O on the concurrent access volume groups).

The cluster lock manager provides two distinct locking models: The CLM locking model and the UNIX System V locking model. Both of these locking models exist in separate name spaces and do not interact. The usage of these locking models are provided by an application programming interface (API) made of a set of C language routines, which enables you to acquire, manipulate, and release locks. To use this API, the concurrent applications running on multiple nodes in an HACMP cluster can see and handle the lock resources across the cluster. The shared resources are divided into a lockable entity. A lockable entity is associated with a lock resource.

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 (see Figure 71).

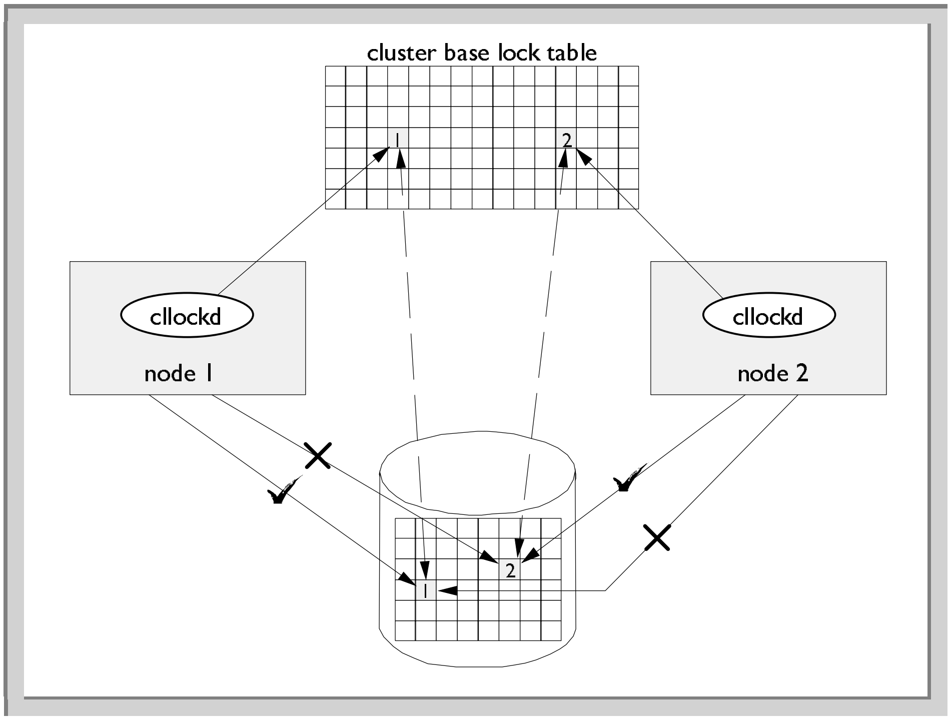


Figure 71. Cluster lock manager

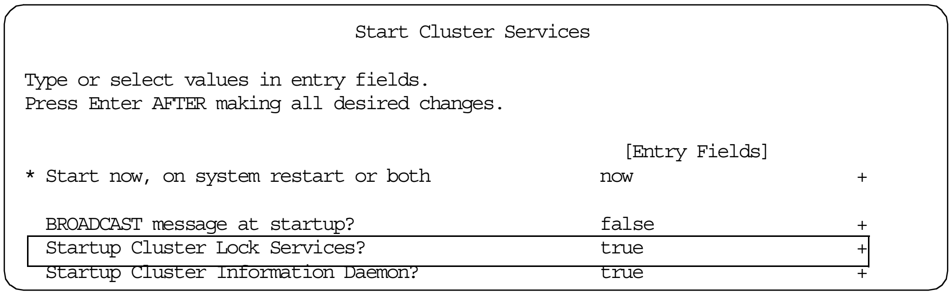
Note that all locks are advisory. The system does not enforce locking. Instead, applications running on the cluster must cooperate to work together. An application that wants to use a shared resource is responsible for first obtaining a lock on that resource, then, attempting to access it.

4.3.2.1 Starting the cluster lock manager

To start the cluster lock manager, you have to specify the necessary option during the cluster manager startup. You can use the following SMIT panel for this purpose:

```
root@goofy:/ # smit -C hacmp
        Cluster Services
                Start Cluster Services
```

Specify *true* as Startup Cluster Lock Services (shown in the rectangle of the following SMIT panel). Then press **Enter**.



```
                           Start Cluster Services

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* Start now, on system restart or both                  now                 +

  BROADCAST message at startup?                         false               +
  Startup Cluster Lock Services?                        true                +
  Startup Cluster Information Daemon?                   true                +
```

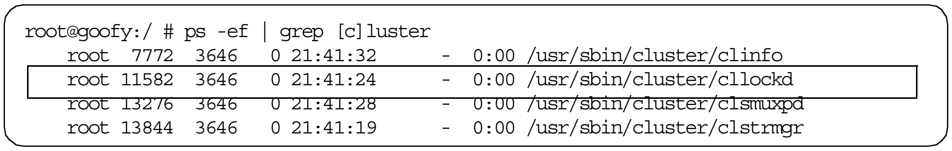
If you prefer the command line interface:

```
root@goofy: / #/usr/sbin/cluster/etc/rc.cluster -boot -N -l -i
```

Then, you will see the following start up messages:

```
0513-059 The clstrmgr Subsystem has been started. Subsystem PID is 13844.
0513-059 The cllockd Subsystem has been started. Subsystem PID is 11582.
0513-059 The clsmuxpd Subsystem has been started. Subsystem PID is 13276.
0513-059 The clinfo Subsystem has been started. Subsystem PID is 7772.
```

You can check the existence of the cllockd daemon process as shown in the rectangle:



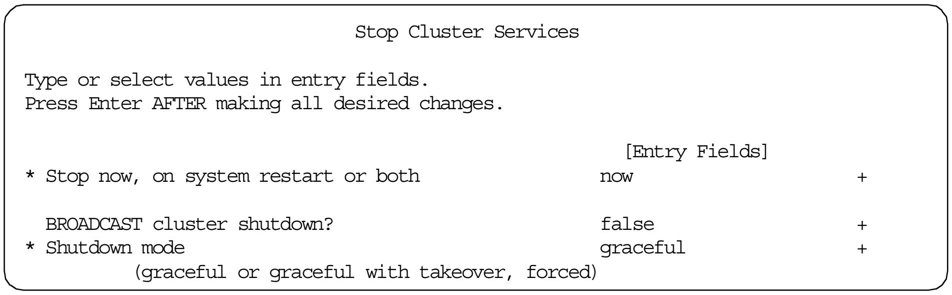
```
root@goofy:/ # ps -ef | grep [c]luster
    root  7772  3646   0 21:41:32      -  0:00 /usr/sbin/cluster/clinfo
    root 11582  3646   0 21:41:24      -  0:00 /usr/sbin/cluster/cllockd
    root 13276  3646   0 21:41:28      -  0:00 /usr/sbin/cluster/clsmuxpd
    root 13844  3646   0 21:41:19      -  0:00 /usr/sbin/cluster/clstrmgr
```

4.3.2.2 Stopping the cluster lock manager

To stop the cluster lock manager, you have to specify the necessary option during the cluster manager shutdown. You can use the following SMIT panel for this purpose.

```
root@goofy:/ # smit -C hacmp
        Cluster Services
                Stop Cluster Services
```

Then press enter twice.



```
                              Stop Cluster Services

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                    [Entry Fields]
* Stop now, on system restart or both              now                    +

  BROADCAST cluster shutdown?                      false                  +
* Shutdown mode                                    graceful               +
          (graceful or graceful with takeover, forced)
```

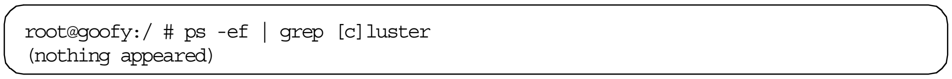
If you prefer the command line interface:

```
root@goofy: / # /usr/sbin/cluster/utilities/clstop -y -N -s -g
```

Then you will see the following shutdown messages:

```
0513-044 The stop of the clstrmgr Subsystem was completed successfully.
0513-044 The stop of the clsmuxpd Subsystem was completed successfully.
0513-044 The stop of the clinfo Subsystem was completed successfully.
```

You can again check the non-existence of the `cllockd` daemon process:



```
root@goofy:/ # ps -ef | grep [c]luster
(nothing appeared)
```

4.3.3 Shared disk solution on an RS/6000 SP system

On an RS/6000 SP system, you can select the following shared disk environment solutions. They are based on completely different technology, but can emulate the concurrent access environment.

- VSD (Virtual Shared Disk)
- RVSD (Recoverable Virtual Shared Disk)
- GPFS (General Parallel File System)

The VSD supplies a pseudo device layer on top of the LVM. If the target disk devices are attached to the local node, each I/O is simply passed to the LVM. If the target disk devices are attached to a remote node, each I/O is sent to the remote node's VSD using the UDP protocol, then handled in the remote node and returned using the same communication path (see Figure 72 on page 227). To achieve high I/O bandwidth, the VSD exploits the high speed inter-connect switch (SP Switch) that is a feature of the RS/6000 SP system.

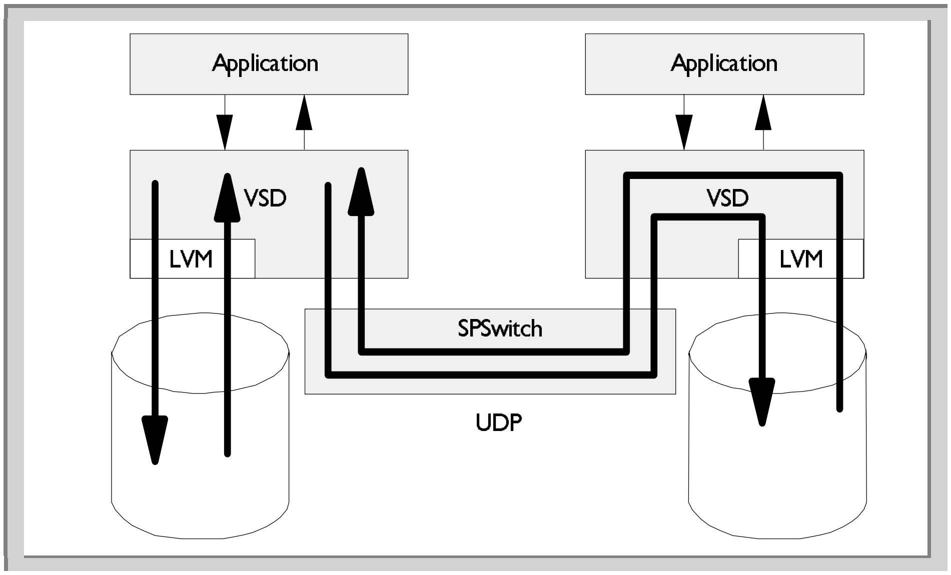


Figure 72. Concept of the Virtual Shared Disk

The VSD device appeared as a raw I/O type device, such as a raw logical volume. From the application point of view, there is no distinct difference between the VSD and the logical volume device.

The RVSD is a separate daemon subsystem named hc that handles the failure situation of the remote node, since, in the shared disk environment provided by the VSD on RS/6000 SP systems, only one node failure can cause the stop of the service. The RVSD is running on each nodes among the VSD configuration, and sends heartbeat packets to each other. If a predefined number of heartbeat packets are dropped, then the hc daemon initiates the take over of the shared disk devices like the HACMP cluster manager daemon (see Figure 73 on page 228). Therefore, the use of RVSD is highly recommended in a VSD configuration.

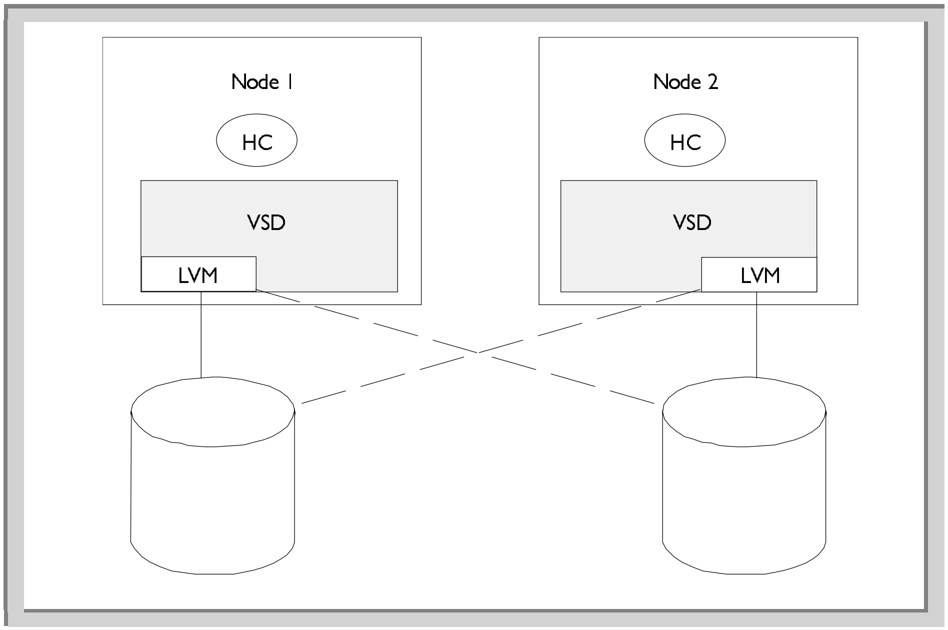


Figure 73. Concept of the Recoverable Virtual Shared Disk

Prior to PSSP Version 3.1, the VSD was a function of the PSSP, although the RVSD was a separately orderable priced feature of PSSP. In PSSP Version 3.1, these two functions are combined together, so it is not necessary anymore to order them separately.

The GPFS is a separately ordered software product that enables the file system access on top of VSD devices among the nodes of the SP system. The file system created on the GPFS file server node (also a VSD/RVSD server and/or client node) can be mounted by a GPFS client node, and appears like a local attached file system.

For further technical detailed information about these software function and products, please consult the following publications and redbooks.

- *PSSP for AIX Version 3.1 Managing Shared Disks*, SA22-7349
- *RS/6000 SP System Management Guide*, SG24-5628
- *GPFS: A Parallel file system*, SG24-5165

4.3.4 What is the HC daemon?

The HC daemon is included in the fileset cluster.hc.rte for HACMP, or cluster.es.hc.rte for HACMP/ES, and provides a way to send information and keep the heartbeat between members of a cluster. The reason HACMP/CRM provides this daemon is to provide the equivalent functional support for the Oracle Parallel Server (also known as OPS). This fact is officially documented in *HACMP Version 4.3 AIX:Programming Locking Applications*, SC23-4281. So, if you do not use the OPS on the RS/6000 cluster systems, then you don't need, and should not install, this fileset.

The hc program is installed under the /usr/lpp/csd directory. The RVSD is also installed under the /usr/lpp/csd directory. As stated in the /usr/sbin/cluster/hc/README file, the cluster.hc.rte and cluster.es.hc.rte filesets are mutually exclusive.

```
root@mickey:/ # lslpp -f cluster.hc.rte
  Fileset               File
  ----------------------------------------------------------------------------
Path: /usr/lib/objrepos
  cluster.hc.rte 4.3.1.0
                        /usr/sbin/cluster/hc
                        /usr/lpp/csd/bin
                        /usr/lpp/csd/bin/hc_members -> /usr/sbin/cluster/hc/hc_members
                        /usr/sbin/cluster/hc/hc_members
                        /usr/sbin/cluster/hc/hc.h
                        /usr/sbin/cluster/hc/hc
                        /usr/sbin/cluster/hc/hacmp_hc_start
                        /usr/lpp/csd/bin/hacmp_vsd_up1 ->
/usr/sbin/cluster/hc/hacmp_vsd_up1
                        /usr/lpp/csd/bin/hacmp_vsd_up2 -> /usr/lpp/csd/bin/hacmp_vsd_up1
                        /usr/lpp/csd/bin/hc -> /usr/sbin/cluster/hc/hc
                        /usr/lpp/csd/bin/hacmp_hc_start ->
/usr/sbin/cluster/hc/hacmp_hc_start
                        /usr/lpp/csd
                        /usr/sbin/cluster/hc/README
                       /usr/lpp/csd/bin/hacmp_vsd_down1 -> /usr/lpp/csd/bin/hacmp_vsd_up1
                       /usr/lpp/csd/bin/hacmp_vsd_down2 -> /usr/lpp/csd/bin/hacmp_vsd_up1
                        /usr/sbin/cluster/hc/hacmp_vsd_up1

Path: /etc/objrepos
  cluster.hc.rte 4.3.1.0
                        NONE
```

If you install the cluster.hc.rte fileset, then you will have the following entry in the /etc/inittab file, and the `hc` daemon process is automatically started during the system startup phase.

```
hc:2:respawn:/usr/lpp/csd/bin/hacmp_hc_start # start the hc daemon
```

If you installed the cluster.hc.rte fileset, then you have to make some modification in the /usr/lpp/csd/bin/machines.lst file. It is generated by the HC daemon at startup, and should be updated with service IP addresses of the cluster nodes.

```
root@mickey:/ # ls -l /usr/lpp/csd/bin/machines.lst
-rw-r--r--   1 root     system        14 Sep 22 15:22 /usr/lpp/csd/bin/machines.lst
```

By default, it contains only the loopback address definition.

```
root@mickey:/ # cat /usr/lpp/csd/bin/machines.lst
1 127.0.0.1 0
```

Without the necessary configuration, the startup of the cluster manager will fail with the following message (from the /tmp/hacmp.out file):

```
Sep 22 15:27:27 EVENT START: node_up_complete mickey

+ + basename /usr/sbin/cluster/events/node_up_complete
PROGNAME=node_up_complete
+ + cl_get_path
HA_DIR=
+ NODENAME=mickey
+ VSD_CMD=/usr/lpp/csd/bin/hacmp_vsd_up2
+ STATUS=0
+ [ ! -n  ]
+ EMULATE=REAL
(many lines are snipped off)
hacmp_vsd_up2[360] grep -v grep
hacmp_vsd_up2[360] grep /usr/lpp/csd/bin/hc
hacmp_vsd_up2[360] awk {print $2}
HCPID=12386
hacmp_vsd_up2[361] [ -n 12386 -a 12386 != -1 ]
hacmp_vsd_up2[362] break
hacmp_vsd_up2[374] sleep 2
hacmp_vsd_up2[374] [[ -n  1 1 1 1 1 1 1 1 ]]
hacmp_vsd_up2[377] hc_members 1 1 1 1 1 1 1 1
/usr/lpp/csd/bin/hacmp_vsd_up2[377]: hc_members:  not found.
hacmp_vsd_up2[378] [ 127 -ne 0 ]
hacmp_vsd_up2[379] echo hacmp_vsd_up2: Error passing Membership  1 1 1 1 1 1 1 1
 to HC
hacmp_vsd_up2: Error passing Membership  1 1 1 1 1 1 1 1 to HC
hacmp_vsd_up2[380] exit 1
+ [ 1 -ne 0 ]
+ exit 1
Sep 22 15:27:33 EVENT FAILED:1: node_up_complete mickey
```

To avoid this erroneous situation, you have to update this file using the following description.

- N   ADDR   0

  N   Node number of the cluster nodes.

  ADDR   Service addresses of the cluster nodes used by the heartbeat interchange.

  0   Instance number (not used).

There are two APARs that address this problem, IX83325 for HACMP Version 4.2.2 and IX85575 for HACMP Version 4.3, but they were not finished when this book was written.

4.4 History of the concurrent access

Historically, the CLVM is a part of the HACMP product. Prior to AIX Version 4.1, users essentially had to have two separate versions of the LVM on their systems (one for non-concurrent provided by the base AIX operating system and one concurrent provided by HACMP/CRM). In AIX Version 4.1, there is a separate fileset introduced named prpq.clvm, where the functionality of the CLVM code has been integrated into the regular LVM code. This allows users to run the standard non-concurrent volume groups and concurrent logical volume groups on the same system. Most users, however, will not notice the difference except for some more options in the LVM.

This package was called prpq.clvm to emphasize that the user benefits from a PRPQ (Program Request for Pricing Quotation) product (program number 5799-FTZ), which is subject to very specific levels of support. After this product has been put onto a system, the old LVM fileset (bos.rte.lvm) is removed. Thus only prpq.clvm fixes will go on the system. Additionally, the prpq.clvm cannot be rejected or taken off the system. There is no separate fileset CLVM after AIX Version 4.2.0.

In the logical volume manager prior to AIX Version 4.3.0, there was a restriction on volume groups running in concurrent mode. If a volume group was running in concurrent mode, then no action that constituted an addition was allowed. As a concurrent volume group is one that is shared between two or more RS/6000s, then the complication was to keep each version of the ODM consistent with respect to the shared volume group. It was a design issue on how to correctly propagate changes in the ODM to the other machines that share the volume group.

With AIX Version 4.3.0, you can now create logical volumes while the volume group is varied on in concurrent mode. In earlier versions, you had to vary off the volume group on all systems sharing this volume group, create a logical volume with one node, then import that concurrent volume group into the other nodes and see if there was a name clash. If there was a name clash, then the import of the concurrent volume group would fail. In the new method, when you try to create a concurrent logical volume while the volume group is varied on in concurrent mode, there is a very complex negotiation that is attempted. The creating node sends a signal and asks all the nodes that are registered for permission (this registration was done during the concurrent varyon time).

Then, each AIX system will search its ODM and see if that name already exists. Note that it does not just search the ODM for that name within the concurrent volume group, it searches for that name in all volume groups.

While one machine might not have lvxx as a pre-existing logical volume, another machine might. If all the nodes, or at least those that can respond, come back with an okay for that possible name, then the concurrent logical volume is created and all the nodes sharing the logical volumes have their ODMs updated. Implied with this update is the fact that a special file, /dev/lvxx, is also created on each concurrent node.

But what happens if there is a node that was off-line and that node happens to have a logical volume that will clash with the newly created logical volume? As stated in the previous paragraph, the node creating the logical volume will wait for a reasonable time for all the nodes to respond. If a node stayed off-line that whole time, it would not get a chance to reply and disagree with a name addition to the shared volume group. So, a restriction was added to the varyonvg code. If during varyonvg of a shared volume group a name conflict between the node varying on and the shared volume group is detected, the node performing the `varyonvg -c` will fail with a message that a name conflict has been detected. It will then be up to the administrator to rename the clashing logical volume or go to one of the legal nodes and remove the logical volume.

The removal of a concurrent logical volume is also allowed now. The logic behind this is fairly simple, except a poll is done on all the shared nodes to make sure that this shared logical volume is not in an open state. The open state indicates that some program or file system has it open and in use. If any node has the logical volume open, a concurrent rmlv will fail.

Chapter 5. The AIX journaled file system

This chapter is dedicated to the AIX journaled file system (also referred to as JFS in this chapter). The JFS is not really an integral part of the Logical Volume Manager, but it is certainly one of the applications that uses it the most. The chapter will provide some knowledge about the way the AIX JFS file system works and how it is constructed.

5.1 What is a journaled file system?

A file system is a hierarchical structure (file tree) of files and directories. This type of structure resembles an inverted tree with the root at the top and branches at the bottom. The branches represent the directories and the leaves represents files. This file tree uses directories to organize data and programs into groups, allowing the management of several directories and files at one time. You can store all kinds of data in a file system. Figure 74 shows you an example of a directory structure.

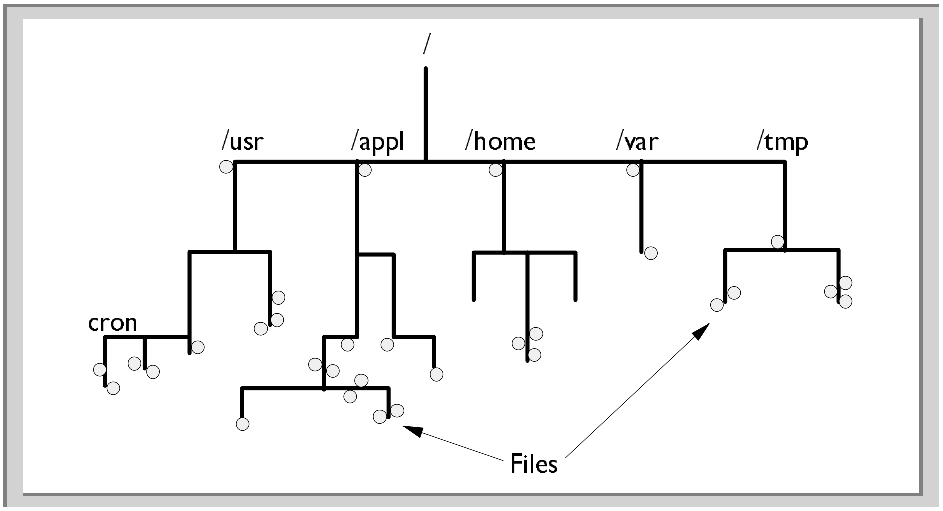


Figure 74. A file system structure

A file system is a set of files, directories, and other structures. File systems maintain information and identify where a file or directory's data is located on the disk. Besides files and directories, file systems consist of:

- The superblock
- The i-nodes

- The data blocks
- The allocation bitmaps

Journaled file systems are created on top of a logical volume.

5.2 The JFS structure

This section describes the components of a file system. We have seen that the parts that would be of interest to the users would be the directories and files, but other elements are present to make the retrieval of information possible and efficient.

5.2.1 The superblock

The superblock maintains information about the entire file system. The superblock is 4096 bytes in size and starts at byte offset 4096 on the logical volume it includes the following fields:

- Size of the file system
- Number of data blocks in the file system
- A flag indicating the state of the file system
- Allocation group sizes

The `dumpfs` command shows you the superblock as well as the i-node map, and disk map information for the file system or special device specified. An example of the `dumpfs` command is shown below.

```
brenzef[root] # dumpfs /dev/hd4

/dev/hd4:
magic                   0x65872143      cpu type                0x0
file system type        0               file system version     1
file system size        65536           fragment size           4096
block size             4096            allocation group size   2048 (frags)
inodes/allocation grp   4096            compress                0
file system name        /               volume name             root
log device              0xa0003         log serial number       0x29
file system state       1               read only               0
last change             Tue Aug 17 18:37:17 CDT 1999
```

At the end of the chapter, you should be comfortable enough with the JFS to understand each of the fields presented in the example above.

5.2.2 Logical blocks

A logical block contains a file or directory data. These units are 4096 bytes in size. Logical blocks are not tangible entities; however, the data in a logical block consumes physical storage space on the disk. Each file or directory consists of 0 or more logical blocks. Fragments, as opposed to logical blocks, are the basic units for allocated disk space in the journaled file system (JFS). Each logical block allocates fragments for the storage of its data.

5.2.3 Disk i-nodes

Each file and directory has an i-node that contains access information such as file type, access permissions, owner's user ID and group ID (UID & GID), and number of links to that file. These i-nodes also contain addresses for finding the location on the disk where the data for a logical block is stored.

Each i-node has an array of numbered sections. Each section contains an address for one of the file or directory's logical blocks. These addresses indicate the starting fragment and the total number of fragments included in a single allocation. For example, a file with a size of 4096 bytes has a single address on the i-node's array. Its 4096 bytes of data are contained in a single logical block. A larger file with a size of 6144 bytes has two addresses. One address contains the first 4096 bytes and a second address contains the remaining 2048 bytes (a partial logical block). If a file has a large number of logical blocks, the i-node does not contain the disk addresses. Instead, the i-node points to an indirect block that contains the additional addresses. Indirect blocks are discussed in Section 5.2.5.2, “Single indirect addressing” on page 239, and in Section 5.2.5.3, “Double indirect addressing” on page 240.

The number of disk i-nodes available to a file system depends on the size of the file system, the allocation group size (8 MB by default), and the ratio of bytes per i-node (4096 by default). These parameters are given to the `mkfs` command at file system creation. When enough files have been created to use all the available i-nodes, no more files can be created, even if the file system has free space. The number of available i-nodes can be determined by using the `df -v` command. Disk i-nodes are defined in the `/usr/include/jfs/ino.h` file. A sample of the `df -v` command would be:

```
brenzef[root] # df -v
Filesystem    1024-blocks  Used      Free %Used  Iused    Ifree %Iused Mounted on
/dev/hd4        32768     25484      7284   78%   1459    14925     9% /
/dev/hd2       737280    691064     46216   94%  23552   160768    13% /usr
/dev/hd9var     16384      6552      9832   40%    314     3782     8% /var
/dev/hd3        49152      7608     41544   16%     49    12239     1% /tmp
/dev/lv1      1638400     52320   1586080    4%     18   409582     1% /ronal
```

The sixth column gives the number of i-nodes used and the eighth column gives the percentage.

5.2.4 Disk i-node structure

Each disk i-node in the journaled file system is a 128-byte structure. The offset of a particular i-node within the i-node list of the file system produces the unique number (i-number) by which the operating system identifies the i-node. A bit map, known as the i-node map, tracks the availability of free disk i-nodes for the file system. Disk i-nodes include the following information:

Field	Contents
i_mode	Type of the file and access permission mode bits
i_size	Size of the file in bytes
i_uid	Access permissions for the user
i_gid	Access permissions for the group
i_nblocks	Number of blocks allocated to the file
i_mtime	Last time the file was modified
i_atime	Last time the file was accessed
i_ctime	Last time the i-node was modified
i_nlink	Number of hard links to the file
i_rdaddr[8]	Real disk addresses of the data
i_rindirect	Real disk address of the indirect block, if any

It is impossible to change the data of a file without changing the i-node, but it is possible to change the i-node without changing the contents of the file. For example, when permission is changed, the information within the i-node (i_ctime) is modified, but the data in the file remains the same. An example below shows this. We create a file with the name disk-config by entering the following command:

```
brenzef[root] # df -k >disk-config
```

The file disk-config contains the following information:

```
brenzef[root] # cat disk-config
Filesystem    1024-blocks      Free %Used    Iused %Iused Mounted on
/dev/hd4            32768      6244   81%     1700    11% /
/dev/hd2           753664     18504   98%    25150    14% /usr
/dev/hd9var         16384      9448   43%      316     8% /var
/dev/hd3            49152     22884   54%       51     1% /tmp
/dev/inner         884736    840488    6%       21     1% /inner
/dev/ronald       1015808    885692   13%       65     1% /home/ronald
```

When we run the `istat` command on the file besides the permissions, user and group information, we see when this file was:

- Last updated
- Last modified
- Last accessed

```
brenzef[root] # istat disk-config
Inode 1107 on device 10/4       File
Protection: rw-r--r--
Owner: 0(root)          Group: 0(system)
Link count:   1         Length 454 bytes

Last updated:   Wed Oct 06 12:11:48 1999
Last modified:  Wed Oct 06 12:11:48 1999
Last accessed:  Wed Oct 06 12:11:51 1999
```

When we display the file using the `cat` command and run the `istat` command again we see that the last access time has changed, but not the contents of the file.

```
brenzef[root] # istat disk-config
Inode 1107 on device 10/4       File
Protection: rw-r--r--
Owner: 0(root)          Group: 0(system)
Link count:   1         Length 454 bytes

Last updated:   Wed Oct 06 12:11:48 1999
Last modified:  Wed Oct 06 12:11:48 1999
Last accessed:  Wed Oct 06 12:44:00 1999
```

Disk i-nodes do not contain file or path name information. Directory entries are used to link file names to i-nodes. Any i-node can be linked to many file names by creating additional directory entries with the `link` or `symlink` subroutine. To discover the i-node number assigned to a file, use the `ls -i` command. More on links can be found in Section 5.5.5, “Links” on page 264. The i-nodes that represent files that define devices contain slightly different information from i-nodes for regular files. Files associated with devices are called special files. There are no data block addresses in special device files, but the major and minor device numbers are included in the i_rdev field. For more information on special files, see Section 5.5.3, “Special or device files” on page 260.

In normal situations, a disk i-node is released when the link count (i_nlink) to the i-node equals 0. Links represent the file names associated with the i-node. When the link count is set to 0, all the data blocks associated with the i-node are released to the bit map of free data blocks for the file system. The JFS will mark this i-node number as free in the i-node map.

5.2.5 i-node addressing

The JFS uses the indirect blocks to address the disk space allocated to larger files. Indirect blocks allow greater flexibility for file sizes. The indirect block is assigned using the i_rindirect field of the disk i-node. This field allows three methods for addressing the disk space:

- Direct
- Single indirect
- Double indirect

The exact same methods are used to address disk space in compressed and fragmented file systems.

5.2.5.1 Direct addressing

When the direct method of disk addressing is used, each of the eight addresses listed in the i_rdaddr field of the disk i-node points directly to a disk fragments. The maximum size of a file using the direct method is 8 x 4096 bytes (or 32,768 bytes). The direct disk address method is illustrated in Figure 75 on page 239.

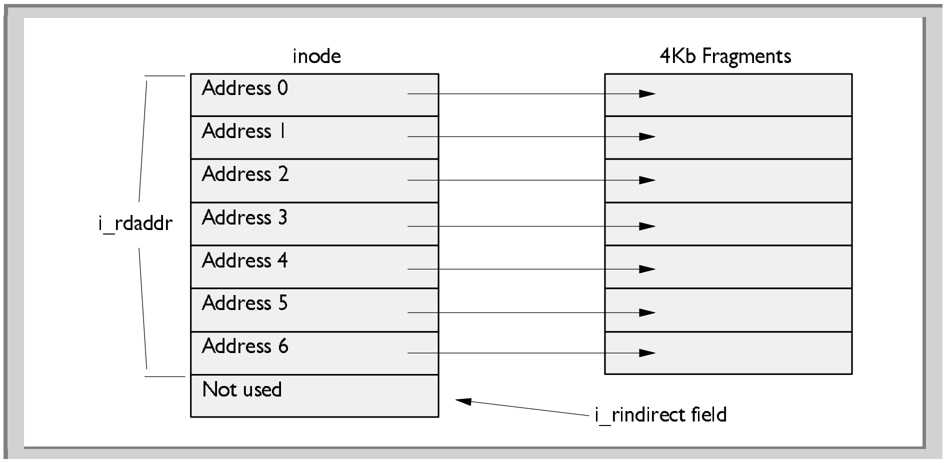


Figure 75. The direct method

5.2.5.2 Single indirect addressing

When a file requires more than 32KB, an indirect block is used to address the file's disk space. The i_rindirect field contains an address that points to either a single indirect block or a double indirect block. When the single indirect disk addressing method is used, the i_rindirect field contains the address of an indirect block containing 1024 addresses. These addresses point to the disk fragments for each allocation. Using the single indirect block geometry, the file can be up to 1024 x 4096 bytes (or 4,194,304 bytes, that is 4 MB). The indirect disk address method is illustrated in Figure 76 on page 240.

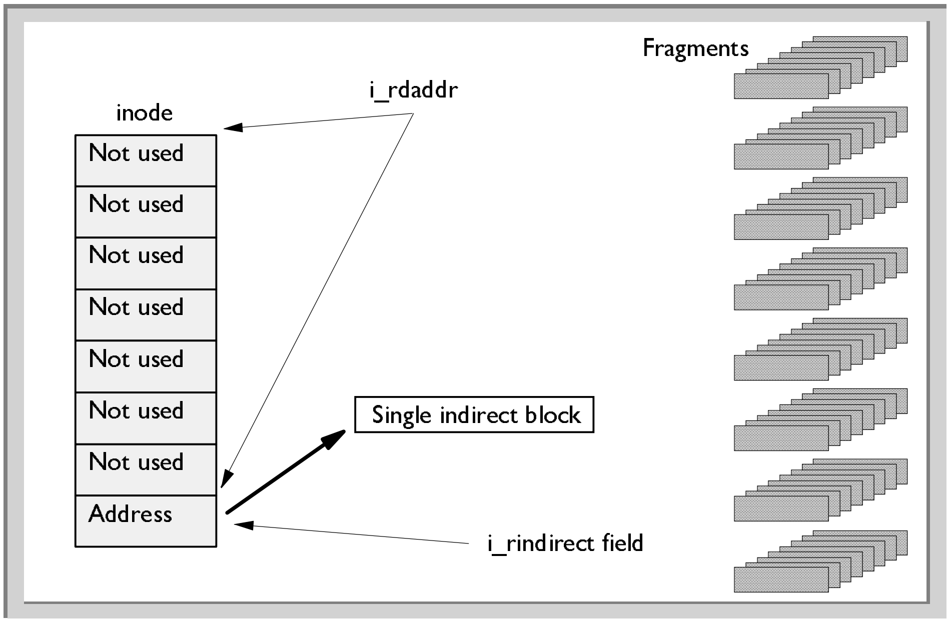


Figure 76. The single indirect method

5.2.5.3 Double indirect addressing

The double indirect addressing method illustrated in Figure 77 on page 241 uses the i_rindirect field to point to a double indirect block. The double indirect block contains 512 addresses that point to indirect blocks, which contain pointers to the fragment allocations. The largest file size that can be used with the double indirect geometry in a file system not enabled for large files is 512 * (1024 x 4096) bytes (or 2,147,483,648 bytes, that is 2 GB).

Beginning in Version 4.2, file systems enabled for large files allow a maximum file size of slightly less than 64 gigabytes (68,589,453,312 bytes). The first single indirect block points to 4096 byte fragments, and all subsequent single indirect blocks point to (32 X 4096) byte fragments. The following produces the maximum file size for file systems enabling large files:

(1 * (1024 * 4096)) + (511 * (1024 * 131072))

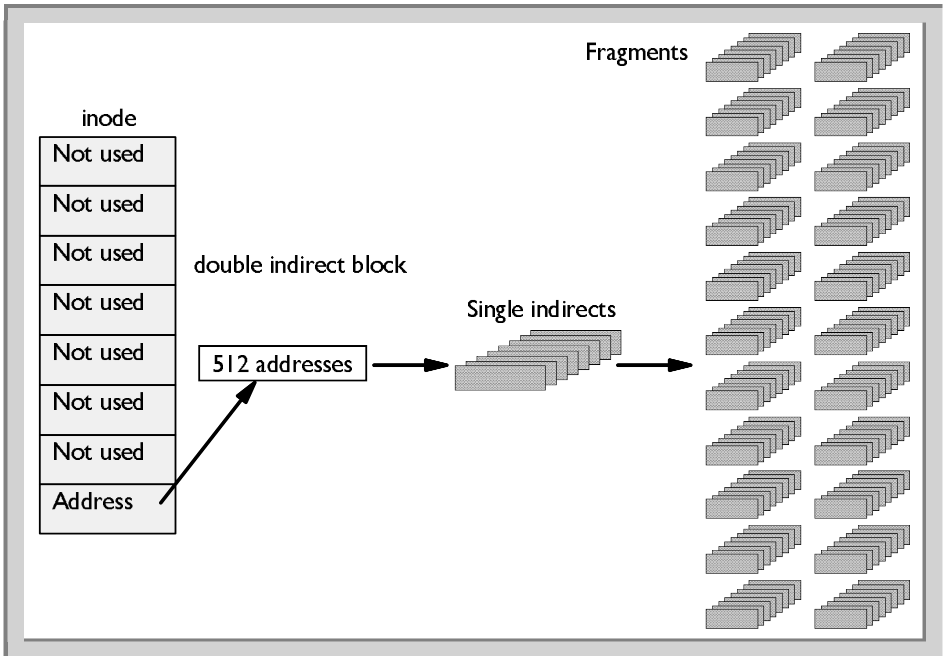


Figure 77. The double indirect method

5.2.6 Fragments

The journaled file system fragment support allows disk space to be divided into allocation units that are smaller than the default size of 4096 bytes. Smaller allocation units or *fragments* minimize wasted disk space by more efficiently storing the data in a file or directory's partial logical blocks. The functional behavior of journaled file system fragment support is based on that provided by Berkeley Software Distribution (BSD) fragment support. Similar to BSD, the JFS fragment support allows users to specify the number of i-nodes that a file system has.

5.2.6.1 Disk utilization

Many UNIX file systems only allocate contiguous disk space in units equal in size to the logical blocks used for the logical division of files and directories. These allocation units are typically referred to as disk blocks and a single disk block is used exclusively to store the data contained within a single logical block of a file or directory.

There are two ways to reduce the number of disk I/O operations:

- Using a relatively large block size

- Maintaining disk block allocation equal in size to the logical block

For example, a file with a size of 4096 bytes or less would be allocated a single 4096 byte disk block if the logical block size is 4096 bytes. A read or write operation would, therefore, only have to perform a single disk I/O operation to access the data on the disk. If the logical block size were smaller, requiring more than one allocation for the same amount of data, then more than one disk I/O operation would be required to access the data. A large logical block and equal disk block size are also advantages for reducing the amount of disk space allocation.

Restricting the disk space allocation unit to the logical block size can, however, lead to wasted disk space in a file system containing numerous files and directories of a small size. Wasted disk space occurs when a a logical block is partially used. The remaining portion remains unused since no other file or directory can write its contents to disk space that has already been allocated. The total amount of wasted disk space can be large for file systems containing a large number of small files and directories.

5.2.6.2 Optimizing disk utilization

In the JFS, however, the disk space allocation unit, referred to as a fragment, can be smaller than the logical block size of 4096 bytes. With the use of fragments smaller than 4096 bytes, the data contained within a partial logical block can be stored more efficiently by using only as many fragments as are required to hold the data. For example, a partial logical block that only has 500 bytes could be allocated a fragment of 512 bytes (assuming a fragment size of 512 bytes), thus greatly reducing the amount of wasted disk space. If the storage requirements of a partial logical block increase, one or more additional fragments will be allocated.

5.2.7 Fragments and number of bytes per i-node (NBPI)

The fragment size for a file system is specified during its creation. The allowable fragment sizes for journaled file systems are 512, 1024, 2048, and 4096 bytes. For consistency with previous versions of AIX, the default fragment size is 4096 bytes. Different file systems can have different fragment sizes, but only one fragment size can be used within a single file system. Different fragment sizes can also coexist on a single system (machine) so that users can select the fragment size most appropriate for each file system. Figure 78 on page 243 shows block and fragments.

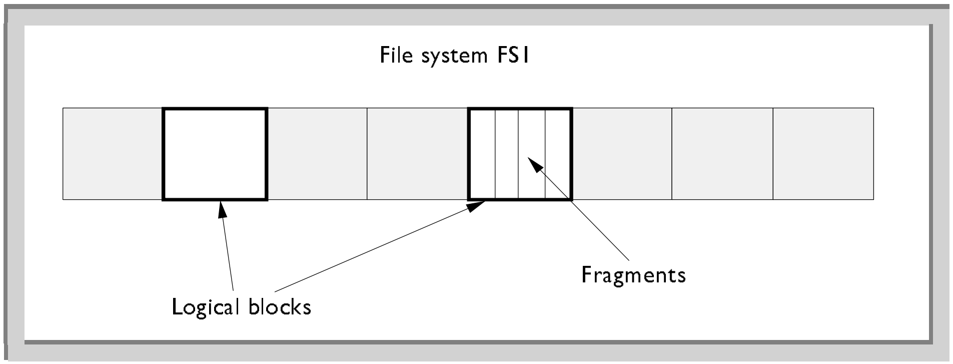


Figure 78. Logical blocks and fragments

The journaled file system fragment support provides a view of the file system as a contiguous series of fragments rather than as a contiguous series of (logical) disk blocks. To maintain the efficiency of disk operations, however, disk space is often allocated in units of 4096 bytes so that the disk blocks or allocation units remain equal in size to the logical blocks. A disk block allocation in this case can be viewed as an allocation of 4096 bytes of contiguous fragments.

Both operational overhead (additional disk seeks, data transfers, and allocation activity) and better utilization of disk space increase as the fragment size for a file system decreases. To maintain the optimum balance between increased overhead and increased usable disk space, the following factors apply to JFS fragment support:

- Disk space allocations of 4096 bytes of fragments are maintained for a file or directory's logical blocks where possible.
- Only partial logical blocks for files or directories less than 32 KB in size can be allocated less than 4096 bytes of fragments.

Maintaining 4096 byte disk space allocations allows disk operations to be more efficient, as previously described in Section 5.2.6.1, “Disk utilization” on page 241.

As the files and directories within a file system grow beyond 32 KB in size, the benefit of maintaining disk space allocations of less than 4096 bytes for partial logical blocks diminishes. The disk space savings as a percentage of total file system space would become negligible, while the extra performance cost of maintaining small disk space allocations remains constant. Since disk space allocations of less than 4096 bytes provide the most effective disk space utilization when used with small files and directories, the logical blocks of files and directories equal to or greater than 32 KB are always allocated 4096 bytes of fragments. Any partial logical block associated with such a large file or directory is also allocated 4096 bytes of fragments.

5.2.7.1 Full and partial logical blocks

A file or directory may contain full or partial logical blocks. A full logical block contains 4096 bytes of data. Partial logical blocks occur when the last logical block of a file or directory contains less than 4096 bytes of data.

For example, a file of 8192 bytes is two logical blocks. The first 4096 bytes reside in the first logical block and the following 4096 bytes reside in the second logical block. Likewise, a file of 6144 bytes consists of two logical blocks. However, the last logical block is a partial logical block containing the last 2048 bytes of the file's data. Only the last logical block of a file can be a partial logical block. Figure 79 shows a file with a size of 4096 bytes that fills one logical block and a file that occupies 6144 bytes, thus six fragments.

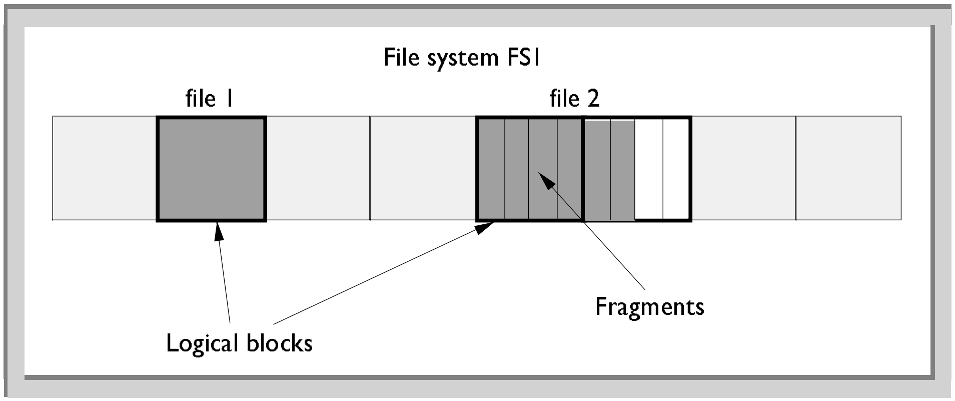


Figure 79. Logical blocks and fragments partially filled

5.2.7.2 Variable number of i-nodes

Since fragment support optimizes disk space utilization, it increases the number of small files and directories that can be stored within a file system. However, disk space is only one of the file system resources required by files and directories; each file or directory also requires a disk i-node. The JFS allows the number of disk i-nodes created within a file system to be specified in case more or fewer than the default number of disk i-nodes is desired. The number of disk i-nodes can be specified at file system creation as the number of bytes per i-node (NBPI). For example, an NBPI value of 1024 causes a disk i-node to be created for every 1024 bytes of file system disk space. Another way to look at this is that a small NBPI value (512 for instance) results in a large number of i-nodes, while a large NBPI value (such as 16,384) results in a small number of i-nodes.

The set of allowable NBPI values vary according to the allocation group size (agsize). The default is 8 MB. In AIX Version 4.1, allocation group size is fixed at 8MB. The allowable NBPI values are 512, 1024, 2048, 4096, 8192, and 16,384, with an allocation group size of 8 MB. More about these values can be found in Figure 81 on page 250.

5.2.7.3 Specifying fragment size and NBPI

Fragment size and the number-of-bytes-per-i-node (NBPI) value are specified during the file system's creation with the `crfs` and `mkfs` commands or by using the System Management Interface Tool (SMIT). The decision of fragment size and how many i-nodes to create for the file system should be based on the projected number of files contained by the file system and their size.

5.2.7.4 Identifying fragment size and NBPI

The file system fragment size and the number-of-bytes-per-i-node (NBPI) value can be identified through the `lsfs` command or the System Management Interface Tool (SMIT). For application programs, the `statfs` subroutine can be used to identify the file system fragment size.

5.2.7.5 Performance costs

Although file systems that use fragments smaller than 4096 bytes as their allocation unit may require substantially less disk space than those using the default allocation unit of 4096 bytes, the use of smaller fragments may incur due to performance costs.

5.2.7.6 Increased allocation activity

Since disk space is allocated in smaller units for a file system with a fragment size other than 4096 bytes, allocation activity may occur more often when files or directories are repeatedly extended in size. For example, a write operation that extends the size of a zero-length file by 512 bytes results in the allocation of one fragment to the file, assuming a fragment size of 512 bytes. If the file size is extended further by another write of 512 bytes, an additional fragment must be allocated to the file. Applying this example to a file system with 4096-byte fragments, disk space allocation would occur only once, as part of the first write operation. No additional allocation activity must be performed as part of the second write operation, since the initial 4096 byte fragment allocation is large enough to hold the data added by the second write operation.

Allocation activity adds performance overhead to file system operations. However, allocation activity can be minimized for file systems with fragment sizes smaller than 4096 bytes if files are extended by 4096 bytes at a time when possible.

5.2.7.7 Free space fragmentation

Using fragments smaller than 4096 bytes may cause greater fragmentation of the disk's free space. For example, consider an area of the disk that is divided into eight fragments of 512 bytes each. Suppose that different files, requiring 512 bytes each, have written to the first, fourth, fifth, and seventh fragments in this area of the disk, leaving the second, third, sixth, and eighth fragments free. Although four fragments representing 2048 bytes of disk space are free, no partial logical block requiring four fragments (or 2048 bytes) will be allocated for these free fragments, since the fragments in a single allocation must be contiguous.

Since the fragments allocated for a file or directory's logical blocks must be contiguous, free space fragmentation may cause a file system operation that requests new disk space to fail even though the total amount of available free space is large enough to satisfy the operation. For example, a write operation that extends a zero-length file by one logical block requires 4096 bytes of contiguous disk space to be allocated. If the file system free space is fragmented and consists of 32 non-contiguous 512 byte fragments or a total of 16 KB of free disk space, the write operation will fail, since eight contiguous fragments (or 4096 bytes of contiguous disk space) are not available to satisfy the write operation.

A file system with an unmanageable amount of fragmented free space can be de-fragmented with the `defragfs` command. The execution of `defrags` has an impact on performance. The use of `defragfs` is also described in Section 7.10.1.3, “defragfs” on page 338.

5.2.7.8 Increased fragment allocation map size

More virtual memory and file system disk space may be required to hold fragment allocation maps for file systems with a fragment size smaller than 4096 bytes.

Fragments serve as the basic unit of disk space allocation, and the allocation state of each fragment within a file system is recorded in the file system's fragment allocation map.

5.2.7.9 Journaled file system log size issues

Another size-related issue is the size of the JFS log. In most instances, multiple journaled file systems use a common log configured to be one physical partition in size. For example, after initial installation, all file systems within the root volume group use logical volume hd8 as a common JFS log. When file systems exceed 2 GB or when the total amount of file system space using a single log exceeds 2 GB, the default log size may not be sufficient. In either case, the log sizes should be scaled upward as the file system size increases. The JFS log is limited to a maximum size of 256 MB. A rule of thumb is 2 MB log for 4 GB of journaled file system. This does not mean that you need a 50 MB log logical volume if you have 100 GB of file systems in a particular volume group. It also depends on the amount of write activity performed. If you have 100 GB of file systems, and the applications do not write huge amounts of data, you will not need a big log logical volume.

5.2.8 Allocation bitmaps

Some administration must be done within the file system, because a lot of logical blocks and fragments reside within a file system. That is why two allocation bitmaps exists:

- The fragment allocation map
- The disk i-node allocation

The fragment allocation bitmap records the allocation state of each fragment. The disk i-node bitmap records the status of each i-node.

5.2.9 Allocation groups

The file system space is divided in chunks called allocation groups. Every allocation group contains i-nodes and data blocks. This permits i-nodes and data blocks to be dispersed throughout the file system, and allows file data to lie in closer proximity to its i-node. Despite the fact that the i-nodes are distributed through the disk, a disk i-node can be located using a trivial formula based on the i-number and the allocation group information contained in the super block. The number of allocation groups grow when the file system grows. When an allocation group is added, the file system contains more i-nodes and data blocks.

The first allocation group begins at the start of the file system and contains a reserved area occupying the first 2 x 4096 bytes of the group. The first 4096 bytes of this area are reserved for the logical volume control block (LVCB) block and the second 4096 bytes hold the file system superblock.

Each allocation group contains a static number of contiguous disk i-nodes which occupy some of the group's fragments. These fragments are set aside for the i-nodes at file system creation and extension time. For the first allocation group, the disk i-nodes occupy the fragments immediately following the reserved block area. For subsequent groups, the disk i-nodes are found at the start of each group. See Figure 80. Disk i-nodes are 128 bytes in size and are identified by a unique disk i-node number or i-number. The i-number maps a disk i-node to its location on the disk or to an i-node within its allocation group.

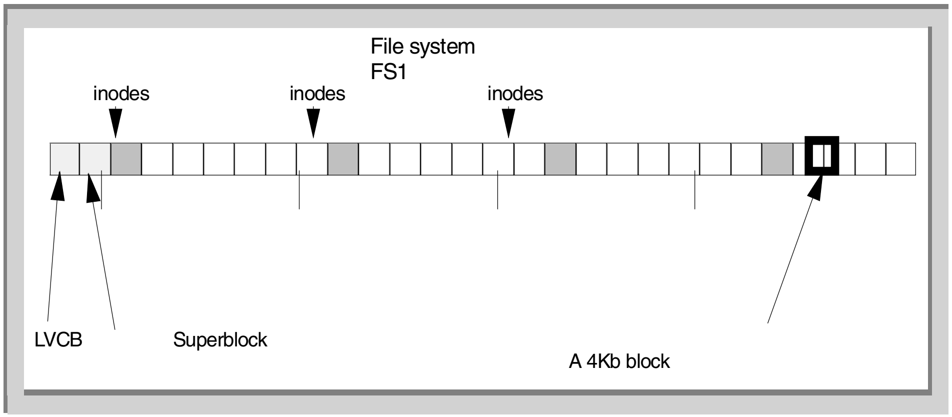


Figure 80. Allocation groups

Note

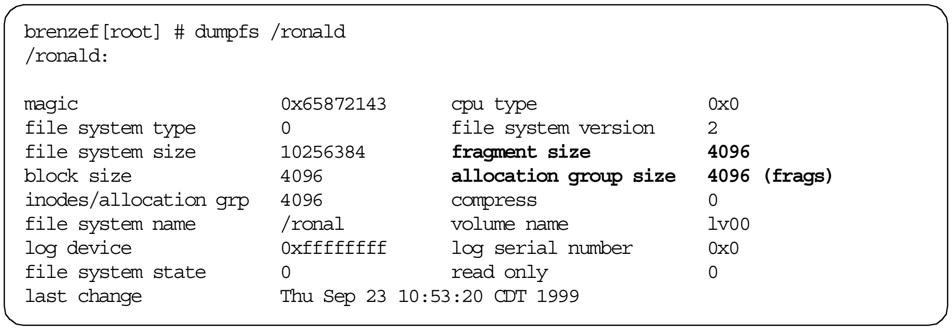
The allocation groups shown in Figure 80 are bigger in a real file system. The groups shown in Figure 80 are meant as an example.

File system's allocation groups are described by three sizes:

- The fragment allocation group size
- The disk i-node allocation group size (specified as the number of fragments)
- Disk i-nodes that exist in each allocation group.

The default allocation group size is 8 MB. Beginning in Version 4.2, it can be as large as 64 MB. These three values are stored in the file system superblock, and they are set at file system creation.

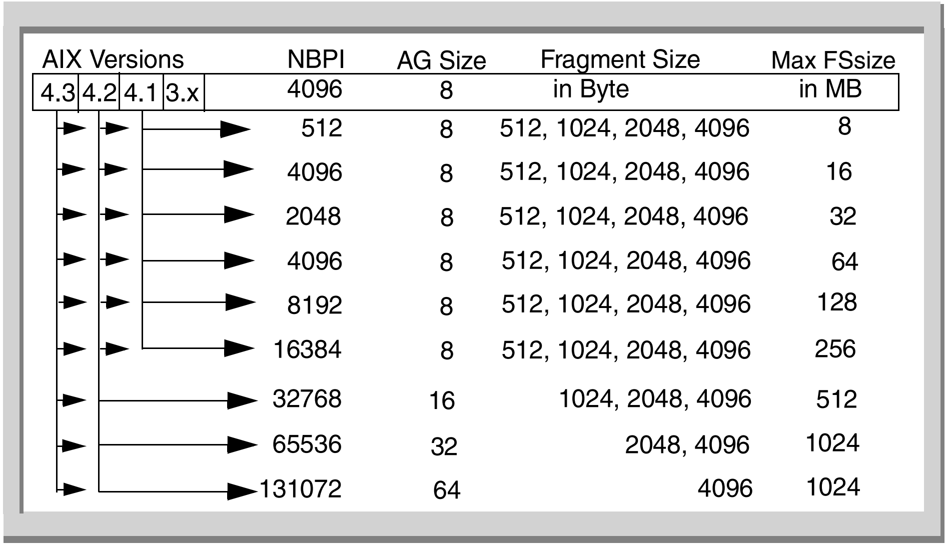
Allocation groups allow the JFS resource allocation policies to use effective methods for achieving good file system I/O performance. These allocation policies try to cluster disk blocks and disk i-nodes for related data to achieve good locality for the disk. Files are often read and written sequentially, and files within a directory are often accessed together. Also, these allocation policies try to distribute unrelated data throughout the file system in an attempt to minimize free space fragmentation.



```
brenzef[root] # dumpfs /ronald
/ronald:

magic                   0x65872143      cpu type                0x0
file system type        0               file system version     2
file system size        10256384        fragment size           4096
block size              4096            allocation group size   4096 (frags)
inodes/allocation grp   4096            compress                0
file system name        /ronal          volume name             lv00
log device              0xffffffff      log serial number       0x0
file system state       0               read only               0
last change             Thu Sep 23 10:53:20 CDT 1999
```

The screen shot above shows a file systems superblock. The superblock shows you the fragment size and the allocation group size (number of fragments) This file system has a allocation group size of 4096 * 4096 =16MB.

AIX Version 4.2 or later supports various allocation group sizes. The JFS segregates file system space into groupings of i-nodes and disk blocks for user data. These groupings are called allocation groups. The allocation group size can be specified when the file system is created. The allocation group sizes are 8 M, 16 M, 32 M, and 64 M. Each allocation group size has an associated NBPI range. The ranges are defined by the following table:



AIX Versions	NBPI	AG Size	Fragment Size in Byte	Max FSsize in MB
4.3 4.2 4.1 3.x	4096	8		
	512	8	512, 1024, 2048, 4096	8
	4096	8	512, 1024, 2048, 4096	16
	2048	8	512, 1024, 2048, 4096	32
	4096	8	512, 1024, 2048, 4096	64
	8192	8	512, 1024, 2048, 4096	128
	16384	8	512, 1024, 2048, 4096	256
	32768	16	1024, 2048, 4096	512
	65536	32	2048, 4096	1024
	131072	64	4096	1024

Figure 81. AIX JFS ranges

The `lsfs` command can be used to evaluate the ranges:

```
brenzef[root] # lsfs -q /ronald-std

Name       Nodename    Mount Pt       VFS  Size    Options    Auto Accounting

/dev/lv01    --       /ronald-std    jfs  32768  rw         no   no

(size: 32768, fs size: 32768, frag size: 4096, nbpi: 4096, compress: no, bf

: false, ag: 8)
```

The fields corresponds with the fields in Figure 81.

When we run the same command on another fileset, notice the different allocation groups size and NBPI.

```
brenzef[root] # lsfs /ronald-large

Name       Nodename    Mount Pt       VFS  Size    Options    Auto Accounting

/dev/lv02    --       /ronald-large  jfs  32768     rw      no   no

(lv size: 32768, fs size: 32768, frag size: 4096, nbpi: 16384, compress:
no, bf: false, ag: 16)
```

5.2.9.1 Maximum journaled file system size

The maximum JFS size is defined when the file system is created. For example, selecting an NBPI ratio of 512 will limit the file system to a size of 8 GB (512 * 2^{24} =8 GB). When creating a JFS file system, the factors listed above (NBPI, fragment size, and allocation group size) need to be weighed carefully. The fileset size limitations can be found in Figure 81 on page 250.

5.2.10 Allocation in compressed file systems

In a file system that supports data compression, directories are allocated disk space. Data compression also applies to regular files and symbolic links whose size is larger than that of their i-nodes.

The allocation of disk space for compressed file systems is the same as that of fragments in fragmented file systems. A logical block is allocated 4096 bytes when it is modified. This allocation guarantees that there will be a place to store the logical block if the data does not compress. The system requires that a write or store operation report an out-of-disk-space condition into a memory-mapped file at a logical block's initial modification. After modification is complete, the logical block is compressed before it is written to a disk. The compressed logical block is then allocated only the number of fragments required for its storage.

In a fragmented file system, only the last logical block of a file (not larger than 32KB) can be allocated less than 4096 bytes. The logical block becomes a partial logical block. In a compressed file system, every logical block can be allocated less than a full block.

A logical block is no longer considered modified after it is written to a disk. Each time a logical block is modified, a full disk block is allocated again according to the system requirements. Reallocation of the initial full block occurs when the logical block of compressed data is successfully written to a disk.

5.2.11 Allocation in file systems with another fragment size

The default fragment size is 4096 bytes. You can specify smaller fragment sizes with the `mkfs` command during a file system's creation. Allowable fragment sizes are: 512, 1024, 2048, and 4096 bytes. You can use only one fragment size in a file system. Read Section 5.2, “The JFS structure” on page 234, for more information on the file system structure.

To maintain efficiency in file system operations, the JFS allocates 4096 bytes of fragment space to files and directories that are 32 KB or larger. A fragment that covers 4096 bytes of disk space is allocated to a full logical block. When data is added to a file or directory, the kernel allocates disk fragments to store the logical blocks. Thus, if the file system's fragment size is 512 bytes, a full logical block is the allocation of 8 fragments.

The kernel allocates disk space so that only the last bytes of data receive a partial block allocation. As the partial block grows beyond the limits of its current allocation, additional fragments are allocated. If the partial block increases to 4096 bytes, the data stored in its fragments are reallocated into 4096-byte allocations. A partial logical block that contains less than 4096 bytes of data is allocated the number of fragments that best matches its storage requirements.

Fragment reallocation also occurs if data is added to logical blocks that represent file holes. A file hole is an empty logical block located prior to the last logical block that stores data. (File holes do not occur within directories.) These empty logical blocks are not allocated fragments. However, as data is added to file holes, allocation occurs. Each logical block that was not previously allocated disk space is allocated 4096 byte of fragment space.

Additional fragment allocation is not required if existing data in the middle of a file or directory is overwritten. The logical block containing the existing data has already been allocated fragments.

JFS tries to maintain contiguous allocation of a file or directory's logical blocks on the disk. Maintaining contiguous allocation lessens seek time because the data for a file or directory can be accessed sequentially and found on the same area of the disk. However, disk fragments for one logical block are not always contiguous to the disk fragments for another logical block. The disk space required for contiguous allocation may not be available if it has already been written to by another file or directory. An allocation for a single logical block, however, always contains contiguous fragments.

The file system uses a bitmap called the fragment allocation map to record the status of every fragment in the file system. When the file system needs to allocate a new fragment, it refers to the fragment allocation map to identify which fragments are available. A fragment can only be allocated to a single file or directory at a time.

5.2.12 Allocation in file systems enabled for large files

In a file system enabled for large files, the JFS allocates two sizes of fragments for regular files.

- A large fragment (32 X 4096) 128 KB
- A 4096 byte fragment

A large block is allocated for logical blocks after the 4 MB boundary; a 4096 byte fragment for logical blocks before the 4 MB boundary.

All non regular files allocate 4096 byte fragments. This geometry allows a maximum file size of slightly less than 64 gigabytes (68.589.453,312 bytes).

A *large* fragment is made up of 32 contiguous 4096 byte fragments. Because of this requirement, it is recommended that file systems enabled for large files have predominantly large files in them. Storing many small files (files less than 4 MB) can cause free-space fragmentation problems. This can cause large allocations to fail with ENOSPC (no space left on device) or EDQUOT (disc quota exceeded) because the file system does not contain 32 contiguous disk addresses.

5.3 How do we use a journaled file system?

Besides the JFS file system, other components are involved when application are using the file system. In this section we focus on these components, like the JFS log, types of files, directory's etc.

5.3.1 The JFS log

AIX uses a special logical volume called the log device as a circular journal for recording modifications to file system meta-data. File system meta-data include the superblock, i-nodes, indirect data pointers, and directories. When meta-data is modified, a duplicate transaction is made to the JFS log. When a `sync()` / `fsync()` occurs, commit records are written to the JFS log to indicate that modified pages in memory have been committed to disk.

Note

The Journaled File System uses a database journaling technique to maintain a consistent file system structure. This involves duplicating transactions that are made to file system meta-data to the JFS log.

The following are examples of when JFS log transactions occur:

- When a file is being created or deleted
- When a `write()` occurs for a file opened with O_SYNC
- When `fsync()` or `sync()` is called
- When a file is opened with O_APPEND
- When a write causes an indirect or double-indirect block to be allocated

The use of a JFS log allows for rapid and clean recovery of file systems if a system goes down. However, there may be a performance trade-off here. If an application is doing synchronous I/O or is creating and/or removing many files in a short amount of time, then there may be a lot of I/O going to the JFS log logical volume. If both the JFS log logical volume and the file system logical volume are on the same physical disk, then this could cause an I/O bottleneck. The recommendation would be to migrate the JFS log device to another physical disk.

Information about I/Os to the JFS log can be recorded using the `filemon` command. If you notice that a file system and its log device are both heavily utilized, it may be better to put each one on a separate physical disk (assuming that there is more than one disk in that volume group). This can be done using the `migratepv` command or via SMIT. For tuning details we refer to Chapter 7, “Performance” on page 303, or the *AIX Performance and Tuning Guide*, SC23-2365.

5.4 File handling

To understand the way files are handled, and to understand the actions started for these operations, a detailed description of the way the journaled file system works is necessary. This section will cover the following topics:

- The system call handler
- The logical file system (LFS)
- the virtual file system (VFS)
- The v-nodes
- The g-nodes
- The i-nodes
- The fragmentation
- The allocation bitmaps and groups

5.4.1 Understanding system call execution

The first layer that is used during file handling is the system call handler, shown in Figure 82 on page 255.

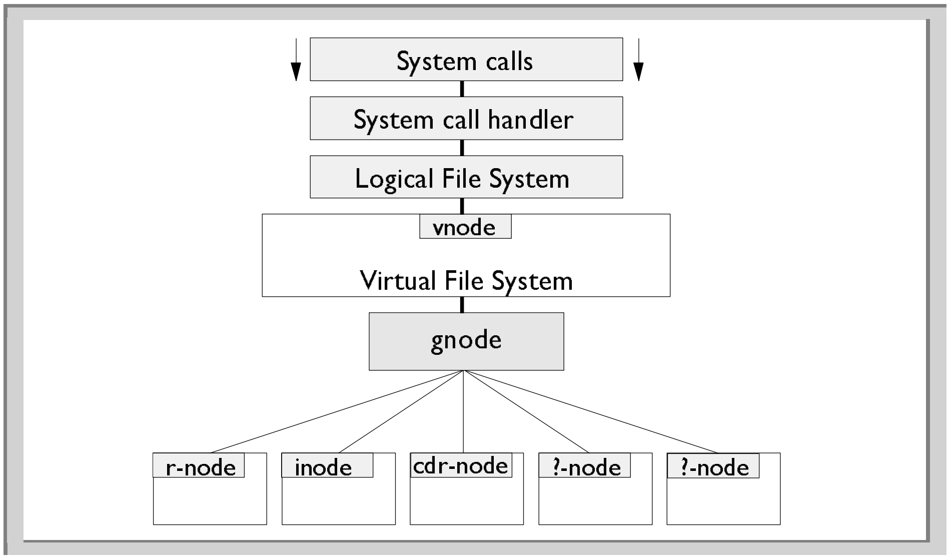


Figure 82. Kernel I/O layers

The system call handler gains control when a user program starts a system call, like `open()`, `close()`, `write()`, etc.. The system call handler changes the protection domain from the caller protection domain, user, to the system call protection domain, kernel.

The system call function returns to the system call handler when it has performed its operation. The system call handler then restores the state of the process and returns to the user program.

5.4.2 The logical file system

The logical file system (LFS) is the level of the file system at which users can request file operations by system call. This level of the file system provides the kernel with a consistent view of what may be multiple physical file systems and multiple file system implementations. As far as the logical file system is concerned, file system types, whether local, remote, or strictly logical, and regardless of implementation, are indistinguishable. The logical file system is located as shown in Figure 82.

A consistent view of file system implementation is made possible by the virtual file system abstraction. This abstraction specifies the set of file system operations that an implementation must include in order to carry out logical file system requests. Physical file systems can differ in how they implement these predefined operations, but they must present a uniform interface to the logical file system. The logical file system and the virtual file system switch support other operating file system access semantics.

Each set of predefined operations implemented constitutes a virtual file system. As such, a single physical file system can appear to the logical file system as one or more separate virtual file systems.

Virtual file system operations are available at the logical file system level through the virtual file system switch. This array contains one entry for each virtual file system, with each entry holding entry point addresses for separate operations. Each file system type has a set of entries in the virtual file system switch.

The logical file system and the virtual file system switch support other operating system file system access semantics. This does not mean that only other operating system file systems can be supported. It does mean, however, that a file system implementation must be designed to fit into the logical file system model. Operations or information requested from a file system implementation need be performed only to the extent possible.

Logical file system can also refer to the tree of known path names in force while the system is running. A virtual file system that is mounted onto the logical file system tree itself becomes part of that tree. In fact, a single virtual file system can be mounted onto the logical file system tree at multiple points, so that nodes in the virtual subtree have multiple names. Multiple mount points allow maximum flexibility when constructing the logical file system view.

5.4.2.1 Component structure of the logical file system

The logical file system is divided into the following components:

- System calls

Implement services exported to users. System calls that carry out file system requests do the following:

  - Map the user's parameters to a file system object. This requires that the system call component use the v-node (virtual node) component to follow the object's path name. In addition, the system call must resolve a file descriptor or establish implicit (mapped) references using the open file component.
  - Verify that a requested operation is applicable to the type of the specified object.

  - Dispatch a request to the file system implementation to perform operations.

- Logical file system file routines

Manage open file table entries and per-process file descriptors. An open file table entry records the authorization of a process's access to a file system object.

A user can refer to an open file table entry through a file descriptor or by accessing the virtual memory to which the file was mapped. The logical file system routines are those kernel services, such as fp_ioctl and fp_select, that begin with the prefix fp_.

- v-nodes0

Provides system calls with a mechanism for local name resolution. Local name resolution allows the logical file system to access multiple file system implementations through a uniform name space.

5.4.3 Virtual file system overview

The virtual file system is an abstraction of a physical file system implementation. It provides a consistent interface to multiple file systems, both local and remote. This consistent interface allows the user to view the directory tree on the running system as a single entity, even when the tree is made up of a number of diverse file system types. The interface also allows the logical file system code in the kernel to operate without regard for the type of file system being accessed.

A virtual file system can also be viewed as a subset of the logical file system tree, that part belonging to a single file system implementation. A virtual file system can be physical (the instantiation of a physical file system), remote (NFS), or strictly logical, like a RAM file system. In the latter case, for example, a virtual file system need not actually be a true file system or entail any underlying physical storage device.

A virtual file system mount point grafts a virtual file system subtree onto the logical file system tree. This mount point ties together a mounted-over v-node (virtual node) and the root of the virtual file system subtree. A mounted-over, or stub, v-node points to a virtual file system, and the mounted VFS points to the v-node it is mounted over.

5.4.4 Understanding virtual nodes (v-nodes)

A virtual node (v-node) represents access to an object within a virtual file system. v-nodes are used only to translate a path name into a generic node (g-node).

A v-node is either created or used again for every reference made to a file by path name. When a user attempts to open or create a file, if the VFS containing the file already has a v-node representing that file, a use count in the v-node is incremented and the existing v-node is used. Otherwise, a new v-node is created.

Every path name known to the logical file system can be associated with, at most, one file system object. However, each file system object can have several names. Multiple names appear in the following cases:

- The object can appear in multiple virtual file systems. This can happen if the object (or an ancestor) is mounted in different virtual file systems using a local file-over-file or directory-over-directory mount.
- The object does not have a unique name within the virtual file system. (The file system implementation can provide synonyms. For example, the use of links causes files to have more than one name. However, the opening of synonymous paths does not cause multiple v-nodes to be created.)

5.4.5 Understanding generic i-nodes (g-nodes)

A generic i-node (g-node) is the representation of an object in a file system implementation. There is a one-to-one correspondence between a g-node and an object in a file system implementation. Each g-node represents an object owned by the file system implementation.

Each file system implementation is responsible for allocating and destroying g-nodes. The g-node then serves as the interface between the logical file system and the file system implementation. Calls to the file system implementation serve as requests to perform an operation on a specific g-node.

A g-node is needed, in addition to the file system i-node, because some file system implementations may not include the concept of an i-node. Thus, the g-node structure substitutes for whatever structure the file system implementation may have used to uniquely identify a file system object.

The logical file system relies on the file system implementation to provide valid data for the following fields in the g-node:

- gn_type - Identifies the type of object represented by the g-node.
- gn_ops - Identifies the set of operations that can be performed on the object.

5.4.6 Understanding the virtual file system interface

Operations that can be performed upon a virtual file system and its underlying objects are divided into two categories:

- Operations upon a file system implementation as a whole (not requiring the existence of an underlying file system object) are called vfs operations.
- Operations upon the underlying file system objects are called v-node (virtual node) operations. Before writing specific virtual file system operations, it is important to note the requirements for a file system implementation.

5.4.7 Accessing files

So, now we have discussed the layered structures from the system call level to the i-node level; so, we can explain what happens when we use open() system call.

The information from the open() system call is passed on to the logical file system (LFS). The logical file system determines if the type of system call is valid for the object. Then, it creates a file table entry for the object and passes the pointer to the virtual file system. The virtual file system creates a v-node for the object and maps it to a g-node. When the object is a NFS file system, an r-node will be mapped to a g-node. When the object belongs to a JFS file system, the object will be mapped to an i-node. When the object is on a CD-ROM file system, a cdrnode will be mapped to the g-node.

5.5 File types

This section describes the type of file you can encounter in the AIX operating system.

5.5.1 Regular files

A regular file is a file that is not in the following categories:

- A special file
- A symbolic link
- A FIFO

- A directory
- A socket

The first bit of the mode field contains the character - . A regular file can, however, be sparse. Sparse files are described in Section 5.5.2, “Sparse files” on page 260. Regular files can be identified by issuing the `ls -l` command and look for the first bit in the mode field. The output should looks similar to the one shown below:

```
brenzef[root] # ls -al employees
total 32
drwxr-xr-x   2 root     system       512 Oct 06 17:02 ./
drwxr-xr-x  32 root     system      4608 Oct 06 17:03 ../
-rw-r--r--   1 root     system        11 Nov 25 17:02 Laurent.Vanel
-rw-r--r--   1 root     system       702 Oct 06 17:06 Hans.Timmers
-rw-r--r--   1 root     system        11 Nov 25 17:02 Ronald.van.der.Knaap
-rw-r--r--   1 root     system        11 Nov 25 17:02 Inge.Wisselink
```

5.5.2 Sparse files

Files that do not have disk blocks allocated for each logical block are called sparse files. Sparse files are created by seeking a file offset and writing data. If the file offsets are greater than 4 MB, then a large disk block of 128 KB is allocated. Applications using sparse files larger than 4 MB may require more disk blocks in a file system enabled for large files than in a regular file system.

5.5.3 Special or device files

A special file is associated with a particular hardware device or other resource of the computer system. The operating system uses special files, sometimes called device files, to provide file I/O access to specific character and block device drivers. Special files, at first glance, appear to be just like ordinary files, in that they:

- Have path names that appear in a directory
- Have the same access protection as ordinary files
- Can be used in almost every way that ordinary files can be used

However, there is an important difference between the two. An ordinary file is a logical grouping of data recorded on disk. A special file, on the other hand, corresponds to a device entity. Examples are:

- An actual device, such as a line printer
- A logical subdevice, such as a large section of the disk drive

- A pseudo device, such as the physical memory of the computer (/dev/mem) or the null file (/dev/null)

Special files are distinguished from other files by having a file type (c or b for character or block) stored in the i-nodes to indicate the type of device access provided. The i-node for the special file also contains the device major and minor numbers assigned to the device at device configuration time.

Several special files are provided with the operating system. By convention, special files are located in the /dev directory or its subdirectories.

5.5.3.1 Character device special files

Character I/O requests are performed by issuing a read or write request on a /dev/xXx character special file for a device, such as /dev/hdisk44 of /dev/rmt0.

Character devices can be identified by issuing the `ls -l` command and looking for the first bit in the mode field. The output should looks similar as the one shown below:

```
brenzef[root] # ls -l /dev/rmt* /dev/tty*
crw-rw-rw-   1 root     system    21,  0 Aug 17 18:37 /dev/rmt0
crw-rw-rw-   1 root     system    21,  1 Aug 17 18:37 /dev/rmt0.1
crw-rw-rw-   1 root     system    21,  2 Aug 17 18:37 /dev/rmt0.2
crw-rw-rw-   1 root     system    21,  3 Aug 17 18:37 /dev/rmt0.3
crw-rw-rw-   1 root     system    21,  4 Aug 17 18:37 /dev/rmt0.4
crw-rw-rw-   1 root     system     1,  0 Oct 06 11:42 /dev/tty
crw-rw--w-   1 uucp     uucp      18,  1 Aug 30 07:37 /dev/tty0
crw-rw-rw-   1 root     system    18,  2 Aug 28 14:59 /dev/tty1
crw-rw-rw-   1 root     system    30,  0 Aug 17 18:55 /dev/ttyp0
crw-rw-rw-   1 root     system    30,  1 Aug 17 18:55 /dev/ttyp1
crw-rw-rw-   1 root     system    30,  2 Aug 17 18:55 /dev/ttyp2
```

The first character in the permission field of `ls` is a `c`, which means that this device is a character device.

Character devices are capable of doing sequential I/O, like disk tape drives and tty devices.

5.5.3.2 Block device special files

Block I/O requests are performed by issuing a read or write on a block special file like /dev/rxxx.

Block devices can be identified by issuing the `ls -l` command and looking for the first bit in the mode field. The output should looks similar as the one shown below:

```
brenzef[root] # ls -l /dev/rhdisk*
      brw-------   1 root     system    14,  5 Aug 30 20:17 /dev/rhdisk0
      brw-------   1 root     system    14,  2 Sep 23 16:16 /dev/rhdisk1
      brw-------   1 root     system    14,  6 Sep 23 16:16 /dev/rhdisk2
      brw-------   1 root     system    14,  4 Sep 23 16:16 /dev/rhdisk3
      brw-------   1 root     system    14,  1 Sep 23 16:16 /dev/rhdisk4
      brw-------   1 root     system    14,  3 Sep 23 16:16 /dev/rhdisk5
```

The first character in the permission field of `ls` is a `b`, which means that this device is a block device.

Block devices can only do random I/O, such as disk devices.

5.5.3.3 Sockets

Sockets were developed in response to the need for sophisticated inter-process facilities to meet the following goals:

- Provide access to communications networks such as the Internet
- Enable communication between unrelated processes residing locally on a single host computer and residing remotely on multiple host machines

Sockets provide a sufficiently general interface to allow network-based applications to be constructed independently of the underlying communication facilities. They also support the construction of distributed programs built on top of communication primitives.

5.5.3.4 FIFO special files

FIFO or first-in-first-out files are unnamed objects created to allow two processes to communicate. One process reads and the other process writes to the FIFO file. This unique type of file is also called a named pipe. The data blocks of the FIFO are manipulated in a circular queue, maintaining read and write pointers internally to preserve the FIFO order of data.

The shell uses unnamed pipes to implement command pipelining. Most unnamed pipes are created by the shell. The | (vertical) symbol represents a pipe between processes.

5.5.3.5 Creating a special file (mknod or mkfifo subroutine)

You can use the mknod and mkfifo subroutines to create new special files. The `mknod` subroutine handles named pipes (FIFO), ordinary, and device files. It creates an i-node for a file identical to that created by the creat subroutine. When you use the `mknod` subroutine, the file-type field is set to indicate the type of file being created. If the file is a block or character-type device file, the names of the major and minor devices are written into the i-node. The mkfifo subroutine is an interface for the mknod subroutine and is used to create named pipes.

The first character in the permission field of `ls` is a p, which means that this device is a FIFO or a named pipe.

5.5.4 Directories

Directories provide a hierarchical structure to the file system and link file and subdirectory names to i-nodes. There is no limit on the depth of nested directories. Disk space is allocated for directories in 4096 byte blocks, but the operating system allocates directory space in 512 byte records.

5.5.4.1 Directory structures

Directories contain a sequence of directory entries. Each directory entry contains three fixed-length fields (the index number associated with the file's i-node, the length of the file name, and the number of bytes for the entry) and one variable length field for the file name. The file name field is null-terminated and padded to 4 bytes. File names can be up to 255 bytes long.

No directory entry is allowed to span 512 byte sections of a directory. When a directory requires more than 512 bytes, another 512 byte record is appended to the original record. If all of the 512 byte records in the allocated data block are filled, an additional data block (4096 bytes) is allotted.

When a file is removed, the space the file occupied in the directory structure is added to the preceding directory entry. The information about the directory remains until a new entry fits into the space vacated.

Every well-formed directory contains the entries . (dot) and .. (dot, dot). The . (dot) directory entry points to the i-node for the directory itself. The .. (dot, dot) directory entry points to the i-node for the parent directory. The mkfs program initializes a file system so that the . (dot) and .. (dot, dot) entries in the new root directory point to the root i-node of the file system.

Access modes for directories have the following meanings:

- Read

Allows a process to read directory entries:

- Write

Allows a process to create new directory entries or remove old ones by using the creat, mknod, link, and unlink subroutines:

- Execute

Allows a process to use the directory as a current working directory or to search below the directory in the file tree.

The example below shows you a directory using the ls command.

```
brenzef[root] # ls -l
total 24
drwxr-xr-x   2 root     system       512 Oct 06 16:53 ./
drwxr-xr-x  32 root     system      4608 Oct 06 16:53 ../
```

This example shows an empty directory. The first bit in the mode field is a d, meaning this is a directory entry.

5.5.5 Links

Links are connections between a file name and an index node reference number (i-node), the internal representation of a file. Because directory entries contain file names paired with i-nodes, every directory entry is a link. The i-node number actually identifies the file, not the file name. By using links, any i-node or file can be known by many different names.

For example, i-node 798 contains a memo regarding November sales in the Amsterdam office. Presently, the directory entry for this memo is:

i-node Number	File Name
798	memo

Because this information relates to information stored in the sales and Amsterdam directories, linking is used to share the information where it is needed. Using the ln command, links are created to these directories. Now the file has three file names:

i-node Number	File Name
798	memo
798	sales/November
798	Amsterdam/november-sales

When you use the pg or cat command to view the contents of any of the three file names, the same information is displayed. You can edit the contents of the i-node from any of the three file. When you use the pg or cat command to view the contents of any of the three file names, the same information is displayed.

If you edit the contents of the i-node from any of the three filenames, the contents of the data displayed by all of the file names will reflect any changes. The contents of the data displayed by all of the file names will reflect any changes.

5.5.5.1 Types of links

Links are created with the `ln` command. There are two kinds of links:

Hard link

Allows access to the data of a file from a new file name. Hard or regular links ensure the existence of a file. When the last hard link is removed, the i-node and its data are deleted.

Note

Hard or regular links can be created only between files that are in the same file system.

Symbolic link

Allows access to data in other file systems from a new file name. The symbolic link is a special type of file that contains a path name. When a process encounters a symbolic link, the process may search that path.

A regular link can be shown with the `ls -li` command. The i-node number for the original file and the link is the same; only the link count is increased. A symbolic link, however, is shown differently when you use the `ls` command. Since symbolic links can reside in other file systems, the i-node number is different. An example of a symbolic link is shown below.

```
brenzef[root] # ls -l link*
-rw-r--r--   2 root     system         0 Oct 06 12:39 link1
lrwxrwxrwx   1 root     system         4 Oct 06 12:39 link2@ -> joop
```

The file `link1` is a regular link that has a link count of 2. The file `link2` is a symbolic link to the file `joop`. The first bit in the mode field is shown as the letter l.

Note

The user who creates a file retains ownership of that file, no matter how many links are created. Only the owner of the file or the root user can set the access mode for that file. However, changes can be made to the file from a linked file name with the proper access mode.

A file or directory exists as long as there is one hard link to the i-node for that file. In the long listing displayed by the `ls -l` command, the number of links to each file and subdirectory is given. All hard links are treated equally by the operating system, regardless of which link was created first.

Chapter 6. Backups

This chapter will not deal with the logic or requirements for a system administrator to back up their system. However, we will deal with the facilities that the LVM provides to make this job easier. We will focus on the new facility brought with AIX Version 4.3.3 that allows the user to perform online backup. Although this is not the goal of this book, we will cover some of the commands used to back up files and volume groups. However, we will not cover the ADSM product. Other redbooks (for example, *Getting Started with ADSM: A Practical Implementation Guide*, SG24-5416) cover these topics in some detail. Similarly, we have not covered sysback.

6.1 Online backups for 4.3.3 and above

The idea of online backup started with customers who realized that if you had a mirrored logical volume, there might be a possibility of using one of the mirrors as a snapshot of the system. The user could use this snapshot, take it off- line, and do a backup of one of the mirrors of the mirrored logical volume. The good and bad thing about LVM is that it is very flexible. It is good because it is designed so that it does not have too many *lockouts* or sections of code that would prevent a clever programmer from making alterations. This type of design allows for constant evolution and design enhancements. The bad thing about this type of design is that some people manipulate LVM in a manner it wasn't designed for. Then, when things blow up, some programmers blame LVM. The logic seems to be that if the code wasn't meant to be manipulated and tweaked, then code to lock out certain actions should be in place. An example is that a majority of commands in LVM are written in shell script. So, the argument goes that if IBM didn't want hackers going in and customizing their copy of LVM, then all commands should be in binaries.

That said, this is the history behind the new online backup feature. First, some customers asked a vendor to design a system where they could take a mirror off-line and back it up. Since the LVM `mklv`, `mklvcopy`, `rmlv`, and `syncvg` commands are all in shell script, it was easy to figure out how to split up a mirrored logical volume. So, some companies were running this script with the incorrect assumption that this was an IBM supported product. Eventually, some customers became quite concerned when they realized that it was not an IBM supported product, but that it was just a very complicated shell script cooked up by some hackers. As a result of this problem, the LVM developers did an analysis of the action of splitting up a mirror and doing an online backup. Note, that an online backup could be a backup of the raw logical volume or a backup of a journaled file system. Taking the whole AIX system in consideration, the LVM team detected some flaws in the original code. Considerations such as major/minor number conflicts, system locking, mirror resynchronization time, and data integrity were not taken into full account in the original code. Some problems were easy to solve through design, but other problems are much more sophisticated and not easily coded out.

As a result of this problem, a new LVM command, `splitlvcopy`, was created. It is an officially supported AIX command. This command is a stopgap measure to replace the homecooked code used by some customers. It has checks that prevent some potential system conflicts, but it still has some limitations in terms of performance and data synchronization. Even after splitlvcopy, there still existed two main problems. The first one is the performance of trying to resync the broken off mirror. The complaint is that if you have two mirrors, A and B, and you take B offline to do a backup, then returning B back to be a partner with A is tedious. Since B has gone away, when it comes back, A thinks that B is a stale mirror; so, the entire mirror must be resynchronized even if nothing has really changed between mirror A and mirror B. And, the main reason that B was split off was so that a snapshot could be taken and so that I/O load on the RS/ 6000 could be lightened. But, the resync of B with A causes the I/O load to go up. The second complaint was that some files (when doing a jfs backup) on the B copy seemed to be missing. This is a very important problem within online backup, and that part of the equation has yet to be solved**.** The problem is metadata of the jfs. To put it in layman's terms, think of metadata as a cloud of information floating above the jfs. This metadata controls file actions and locations. Now there exists windows where some data is still "in the clouds" when mirror B is split off. The metadata has not been completely written to the mirrored logical volume. One method to help get more metadata written to the disk is to use system syncs. However, there is no guarantee that some metadata didn't change between the time the syncs complete and then split off of the mirror. Many customers ask for a guarantee that no data will be lost. The answer is that even with many syncs, we can't guarantee that some user somewhere doesn't alter the metadata right before the splitlvcopy is run. The only way to guarantee that metadata is flushed to the disk and remains unchanged is to close the file system. In other words, you must unmount the file system. However, this is contradictory to the original need for online backup, that is, a method to take a snapshot of the system without bringing down the application running on the mirrored logical volume. Because of this known problem, the following warning is inserted into the splitlvcopy man pages:

Attention: Although the `splitlvcopy` command can split logical volumes that are open, including logical volumes containing mounted file systems, this is not recommended. You may lose consistency between LogicalVolume and

NewLogicalVolume if the logical volume is accessed by multiple processes simultaneously. When splitting an open logical volume, you implicitly accept the risk of potential data loss and data corruption associated with this action. To avoid the potential corruption window, close logical volumes before splitting, and unmount file systems before splitting.

This sounds pretty scary, but it is usually not the case. What typically happens is that during the backup of the system via jfs, the backup program will print out a message that a few files cannot be found. It was decided that this would be equivalent to taking a snapshot of a file system that was having users add and delete files right in the middle of the snapshot. But, as mentioned before, a way to guarantee that the snapshot of the system has true data quiescence, the logical volume should be closed, which implies that the journaled file system be unmounted.

The current modification to online backup does not close the window with the possible in-flight metadata. That is reserved for another, more involved, design. The online backup offered in AIX 4.3 delivers a faster resynchronization method. There are two different options delivered. The first one, with no new special file, was delivered in AIX 4.3.1. The second one, with a new special file name for the backup copy, was introduced in AIX 4.3.2. This gets a little complicated; so, we must first discuss what was available prior to AIX 4.3. In the earlier versions of AIX, a user could write a READX() system call with the options for mirror avoidance placed in the READX() call; so, if one had mirrors A and B, and one wanted to read mirror B, then a programmer would make a system READX() call and put in the option to avoid mirror 1. But, during this time, any writes to copy A will automatically get copied over to copy B. There is no “snapshot” here.

In AIX 4.3.1 when the user runs the `chlvcopy -b` command, no new special file is created for copy B, as it would have been for the `splitlvcopy` command. Instead, the driver is notified that copy B is going to temporarily be a backup logical volume. During this time, all writes to copy A do not get propagated to copy B. But with no new special file, how does a user read from copy B to do the backup? The answer is the `readlvcopy` command, which allows you to simply read from the backup copy when you are reading from the shared /dev special file. This `readlvcopy` is really a glorified READX() call that does the equivalent of a “dd” data dump. But, we stress again, that this method differs from the previously mentioned READX() call by not propagating changes from A to B. Note, one behavior of this option is that after a reboot of the AIX system, the LVM code forgets about that the `chlvcopy` command was ever run. The logical volume mirrors that were temporarily, logically separated, become rejoined. Of course, during the `varyonvg` step of the boot, the stale logical volumes on copy B get reintegrated back into the mirror pair.

In AIX 4.3.2, a temporary new logical volume is created from one of the mirrors, just as in the splitlvcopy command. The difference with this logical volume is that it is smart enough to track changes in the mirrored copies. As in the previous paragraphs, suppose you have a mirror made up of copy A and copy B. The 7chlvcopy -bl <lvname> command is used to split off copy B into a new logical volume. This new logical volume has a special file entry in /dev. The user can then have a program access the data via the /dev special file entry or the readlvcopy command can be used. Now, copy A is considered the “original” and copy B is considered the split off copy. If the user reads from copy A, nothing happens except a normal read. If the user reads from copy B, nothing happens except a normal read. Suppose then, that the user writes to copy A at physical partition 100, then the same partition number on copy B is marked stale. However, the user reading from copy B is allowed to continue reading the stale partition 100. This is a departure from the default LVM mirroring case. Normally, you can’t read from a stale partition, but in this special condition you are allowed to do so. If, for some odd reason, you perform a write on copy B, the partitions on copy B get marked stale but not those on copy A. Put in another way, copy A will never be marked stale during this split operation. However, consider the consequences of writing to copy B. If you do that, then during resync the stale copy will be overwritten by those on copy A and changes you made to copy B will be lost. A major point to this option is that it is persistent across reboots. The -l option will remember that the logical volume mirrors were split off and will still act in that manner after a system reboot. When it comes time to remerge the mirrors, the user runs the chlvcopy -B option to remerge the mirrors. The user will still have to manually run the syncvg command to clear up stale partitions.

But now, the benefit of this new chlvcopy command is that the resync time between copy A and copy B can be very quick. If only a few files changed between A and B, then the resynchronization only has to recopy a few physical partitions from copy A over copy B. However, suppose that every single directory and every single file was somehow changed during the duration of the backup. In this case, there is a possibility that every partition from copy A must now be used to master copy B.

A summation of the three option types is shown:

Table 18. The three possible methods

splitlvcopy	READX()	chlvcopy -b	chlvcopy -b -l <lvname>
Creates a new special file. AIX 4.1, 4.2, 4.3	Reads from common special file via avoidance flags. All versions of AIX	Reads from common special file via READX() or readlvcopy command. AIX 4.3.1 and higher	Creates a new special file. AIX 4.3.2
Persistent across reboots. Very slow resync time.	Not persistent across reboot. No mirror staleness enforcement, no resync.	Not persistent across reboot. Mirror staleness is enforced. No special file for backup persistent across reboot.	Mirror staleness is enforced. Special file does exist for backup.

6.2 Practical examples

We are now going to see three practical examples of:

- A concurrent online mirror backup with AIX Version 4.3.2
- An online JFS backup with AIX Version 4.3.3
- An online backup prior to AIX Version 4.3.2 (the hack)

6.2.1 Concurrent online mirror backup

This was introduced in AIX Version 4.3.2, extending the 4.3.1 online support for raw logical volumes to support:

- Concurrent mode volume groups
- File system and database access

For our three situations, we will consider a volume group, asgard_lv, with a a logical volume, loki_lv, hosting a file system /home/loki. loki_lv is a mirrored logical volumes and has three copies spread across three disk.

```
/home/red # lsvg -l asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    open/syncd    /home/loki
loglv00             jfslog     1     1     1    open/syncd    N/A
```

Here is a list of the mounted file systems. You can see the /home/loki file system at the bottom of the screen.

```
/home/red # df -kI
Filesystem    1024-blocks      Used      Free %Used Mounted on
/dev/hd4             8192      7960       232   98% /
/dev/hd2           466944    444684     22260   96% /usr
/dev/hd9var        122880     82340     40540   68% /var
/dev/hd3            40960     27904     13056   69% /tmp
/dev/hd1            32768     17664     15104   54% /home
/dev/lv00        20004864   3055816  16949048   16% /software
/dev/stripelv       12288       560     11728    5% /stripefs
/dev/loki_lv       163840      6768    157072    5% /home/loki
```

The first step would be to unmount the file system from the mirrored and to run the `chlvcopy` command.

```
/home/red # umount /home/loki
/home/red # chlvcopy -b -c 3 -l loki_bup loki_lv
loki_bup
```

The result of this command is a new logical volume, called loki_bup (specified as argument). We can check that with the `lsvg` command.

```
/home/red # lsvg -l asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    closed/syncd  /home/loki
loglv00             jfslog     1     1     1    closed/syncd  N/A
loki_bup            jfs        0     0     0    closed??????? N/A
```

We now have to use the two different file systems, /home/loki and /home/backup, the read-only version of /home/loki. Let us remount the file system.

```
/home/red # mount /home/loki
/home/red # mount -o ro /dev/loki_bup /home/backup
/home/red # df -kI
Filesystem    1024-blocks      Used      Free %Used Mounted on
/dev/hd4             8192      7960       232   98% /
/dev/hd2           466944    444684     22260   96% /usr
/dev/hd9var        122880     82340     40540   68% /var
/dev/hd3            40960     27904     13056   69% /tmp
/dev/hd1            32768     17664     15104   54% /home
/dev/lv00        20004864   3055816  16949048   16% /software
/dev/stripelv       12288       560     11728    5% /stripefs
/dev/loki_lv       163840      6768    157072    5% /home/loki
/dev/loki_bup      163840      6768    157072    5% /home/backup
```

As you can see, right now the characteristics of the two file systems are similar, with the same size, the same number of i-nodes free, and so on. They also have the same files and directories, Let us change that by creating a new file in the /home/loki file system. Of course, this file shows up in the home/loki file system, but will not be present in our backup.

```
/home/red # ls -l /home/loki
total 3176
-rw-r--r--   1 root     sys      1620000 Jan 08 19:04 file1
drwxrwx---   2 root     system       512 Jan 08 19:03 lost+found
/home/red # ls -l /home/backup
total 3176
-rw-r--r--   1 root     sys      1620000 Jan 08 19:04 file1
drwxrwx---   2 root     system       512 Jan 08 19:03 lost+found
/home/red # echo "blah blah blah blah blah" > /home/loki/new_file
/home/red # ls -l /home/loki
total 3184
-rw-r--r--   1 root     sys      1620000 Jan 08 19:04 file1
drwxrwx---   2 root     system       512 Jan 08 19:03 lost+found
-rw-r--r--   1 root     sys           25 Jan 08 19:29 new_file
{should do fsck here}
/home/red # #{Do backup thingies here}
```

Now that we have finished our backup, it is time to put things back together. First, unmount the /home/backup file system. We can now run the `chlvcopy` command again with the -B option to merge back the separated copy. As you can see from the `lsvg` command, the extra loki_bup logical volume has disappeared.

```
/home/red # umount /home/backup
/home/red # chlvcopy -B loki_lv
Warning, all data contained on logical volume loki_bup will be destroyed.
rmlv: Do you wish to continue? y(es) n(o)? y
rmlv: Logical volume loki_bup is removed.
/home/red # lsvg -l asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    open/stale    /home/loki
loglv00             jfslog     1     1     1    open/syncd    N/A
```

We still have the newly created file, new_file, in our file system.

```
/home/red # ls -l /home/loki
total 3184
-rw-r--r--   1 root     sys      1620000 Jan 08 19:04 file1
drwxrwx---   2 root     system       512 Jan 08 19:03 lost+found
-rw-r--r--   1 root     sys           25 Jan 08 19:29 new_file
/home/red # syncvg -v asgard_vg
/home/red # lsvg -l asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    open/syncd    /home/loki
loglv00             jfslog     1     1     1    open/syncd    N/A
```

6.2.2 The online JFS backup

This version is available with AIX Version 4.3.3. Making an online backup of a mounted JFS file system creates a snapshot of the logical volume that contains the file system while applications are still using the file system. Be aware, though, that since the file writes are asynchronous, the snapshot may not contain all data that was written immediately before the snapshot is taken. Modifications that start after the snapshot begins may not be present in the backup copy. Therefore, it is recommended that file system activity be minimal while the split is taking place.

Split off a mirrored copy

In order to make an online backup of a mounted file system, the logical volume that the file system resides on must be mirrored. The JFS log logical volume for the file system must also be mirrored. The number of copies of the jfs log must be equal to the number of copies of the file system's logical volume.

To split off the mirrored copy of the file system, use the `chfs` command, and you can control which copy is used as the backup by using the copy attribute. The second copy is the default if a copy value is not specified.

The following example shows a copy of the file system /home/loki split off. The example assumes that there are three copies of the file system and three copies of the jfs log, as you can verify by using:

```
/home/red # lsvg -l asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    open/syncd    /home/loki
loglv00             jfslog     1     3     3    open/syncd    N/A
/home/red # df -kI
Filesystem    1024-blocks      Used      Free %Used Mounted on
/dev/hd4             8192      7960       232   98% /
/dev/hd2           466944    444684     22260   96% /usr
/dev/hd9var        122880     82340     40540   68% /var
/dev/hd3            40960     27904     13056   69% /tmp
/dev/hd1            32768     17664     15104   54% /home
/dev/lv00        20004864   3055816  16949048   16% /software
/dev/stripelv       12288       560     11728    5% /stripefs
/dev/loki_lv       163840      6772    157068    5% /home/loki
```

Issuing the `chfs` command, we can use the third copy of /home/loki to produce a new /home/backup file system.

Once this command completes successfully, a copy of the file system is available read-only in /home/backup. Remember that additional changes made to the original file system after the copy is split off are not reflected in the backup copy.

Note

Splitting a mirrored copy of a file systems means that one copy is temporarily dedicated to backup activities and is not available to provide data availability. It is recommended that you have three mirrored copies of the file system so you can recover a disk problem before the backup copy has been reintegrated on the file system.

```
/home/red # chfs -a splitcopy=/home/backup -a copy=3 /home/loki
loki_lvcopy00
backup requested(0x100000)...
log redo processing for /dev/loki_lvcopy00
syncpt record at 3028
end of log 3488
syncpt record at 3028
syncpt address 3028
number of log records  = 12
number of do blocks = 2
number of nodo blocks = 0
```

The /home/backup file system has been created pointing to the third copy of the original file system.

```
/home/red # df -kI
Filesystem    1024-blocks      Used      Free %Used Mounted on
/dev/hd4             8192      7960       232   98% /
/dev/hd2           466944    444684     22260   96% /usr
/dev/hd9var        122880     82340     40540   68% /var
/dev/hd3            40960     27904     13056   69% /tmp
/dev/hd1            32768     17664     15104   54% /home
/dev/lv00        20004864   3055816  16949048   16% /software
/dev/stripelv       12288       560     11728    5% /stripefs
/dev/loki_lv       163840      6772    157068    5% /home/loki
/dev/loki_lvcopy00      163840      6772    157068    5% /home/backup
```

The /home/loki file system is still made by a logical volume with three copies, but now it has stale partitions, since one copy is no longer kept in sync with the other and is made available for read-only access through the /home/backup mount point. Note the new loki_lvcopy00 logical volume; it is in an open state, it cannot be in sync or in stale so that's why there are the question marks and no logical partitions are assigned to it since it only points to a specific copy of lv00.

```
/home/red # lsvg -l asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    open/stale    /home/loki
loglv00             jfslog     1     3     3    open/syncd    N/A
loki_lvcopy00       jfs        0     0     0    open???????   /home/backup
```

The /backup file system has been mounted and is available to use. If you issue the `mount` command, you can see it mounted with read-only permissions and without any jfslog.

Now it is possible to read the data in /backup.

When a copy of a file system is split off, some activity is made on the jfslog in order to keep the structure of the original file system and the read-only copy consistent. During such an operation, no other splitting must be done on the file system that uses the same jfslog. They must be delayed until the previous split is finished.

For example, consider the case of two file systems, /team1 and /team2, that use the same jfslog. If you want to make an online backup of both on the same tape drive, you have to first split /team1, wait for the read-only copy to be available, and then you may split /team2. Finally, you can do the backup. If, for some reason, you split both file systems in the same moment, one split will succeed and the other will fail.

```
/home/red # mount
  node       mounted        mounted over    vfs       date        options
-------- ---------------  ---------------  ------ ------------ ---------------
         /dev/hd4         /                jfs    Dec 31 18:53 rw,log=/dev/hd8
         /dev/hd2         /usr             jfs    Dec 31 18:53 rw,log=/dev/hd8
         /dev/hd9var      /var             jfs    Dec 31 18:53 rw,log=/dev/hd8
         /dev/hd3         /tmp             jfs    Dec 31 18:53 rw,log=/dev/hd8
         /dev/hd1         /home            jfs    Dec 31 18:54 rw,log=/dev/hd8
         /dev/lv00        /software        jfs    Dec 31 18:54 rw,log=/dev/lo
         /dev/stripelv    /stripefs        jfs    Jan 04 23:36 rw,log=/dev/st
         /dev/loki_lv     /home/loki       jfs    Jan 08 19:27 rw,log=/dev/lo
         /dev/loki_lvcopy00 /home/backup     jfs    Jan 08 22:45 ro

/home/red # #{do backup thingies here}
```

Right now, the contents of /home/loki and /home/backup are identical. Let us modify that by creating a new file in /home/loki. As you can see, this new file is not propagated to /home/backup:

```
/home/red # echo "blah blah blah blah blah" > /home/loki/newer_file
/home/red # ls -l /home/loki
total 3192
-rw-r--r--   1 root     sys      1620000 Jan 08 19:04 file1
drwxrwx---   2 root     system       512 Jan 08 19:03 lost+found
-rw-r--r--   1 root     sys           25 Jan 08 19:29 new_file
-rw-r--r--   1 root     sys           25 Jan 08 19:45 newer_file
/home/red # ls -l /home/backup
total 3184
-rw-r--r--   1 root     sys      1620000 Jan 08 19:04 file1
drwxrwx---   2 root     system       512 Jan 08 19:03 lost+found
-rw-r--r--   1 root     sys           25 Jan 08 19:29 new_file
/home/red # lsvg -l asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    open/stale    /home/loki
loglv00             jfslog     1     3     3    open/syncd    N/A
loki_lvcopy00       jfs        0     0     0    open???????   /home/backup
```

It is time now to merge back our separated copy of the logical volume. This step is easy, we just have to umount the /home/backup file system and destroy this file system. The logical volume does not show up anymore in the volume group, and we can check that even the file created while the copies were split is still there.

```
/home/red # umount /home/backup
/home/red # rmfs /home/backup
rmlv: Logical volume loki_lvcopy00 is removed.
/home/red # lsvg -l asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    open/syncd    /home/loki
loglv00             jfslog     1     3     3    open/syncd    N/A
/home/red # ls -l /home/loki
total 3192
-rw-r--r--   1 root     sys      1620000 Jan 08 19:04 file1
drwxrwx---   2 root     system       512 Jan 08 19:03 lost+found
-rw-r--r--   1 root     sys           25 Jan 08 19:29 new_file
-rw-r--r--   1 root     sys           25 Jan 08 19:45 newer_file
```

Reintegrate a mirrored copy.

6.2.3 Online backups prior to 4.3.2 (the hack)

The last method is not officially supported by IBM. The trick here is to reuse the physical partition of one of the copies of a mirror logical volume. Let us see this step by step.

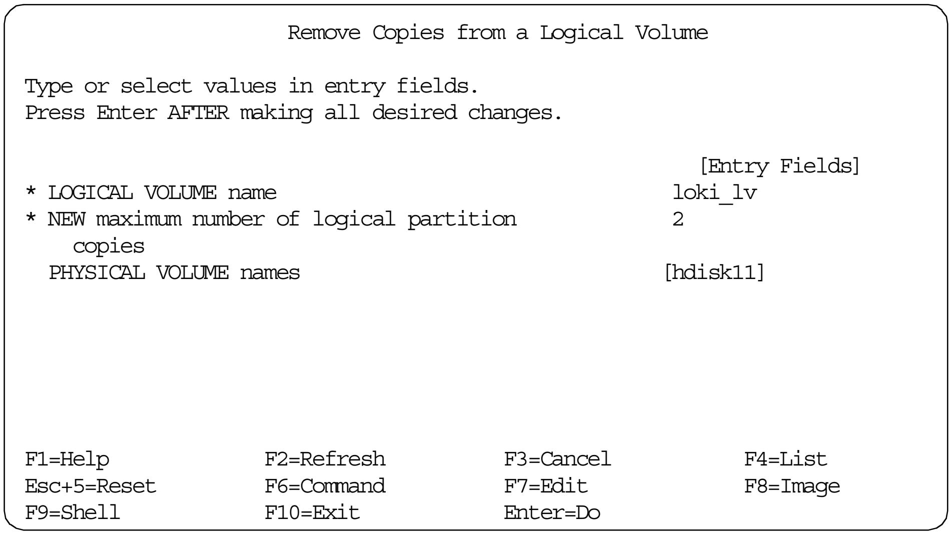
We start with the same volume group, asgard_vg, including loki_lv, hosting /home/loki. loki_lv is mirrored with three copies.

```
/home/red # lsvg -l  asgard_vg
asgard_vg:
LV NAME             TYPE       LPs   PPs   PVs  LV STATE      MOUNT POINT
loki_lv             jfs        20    60    3    open/syncd    /home/loki
loglv00             jfslog     1     3     3    open/syncd    N/A
```

Here is the physical allocation of the loki_lv logical volume. As you can see there are three copies.

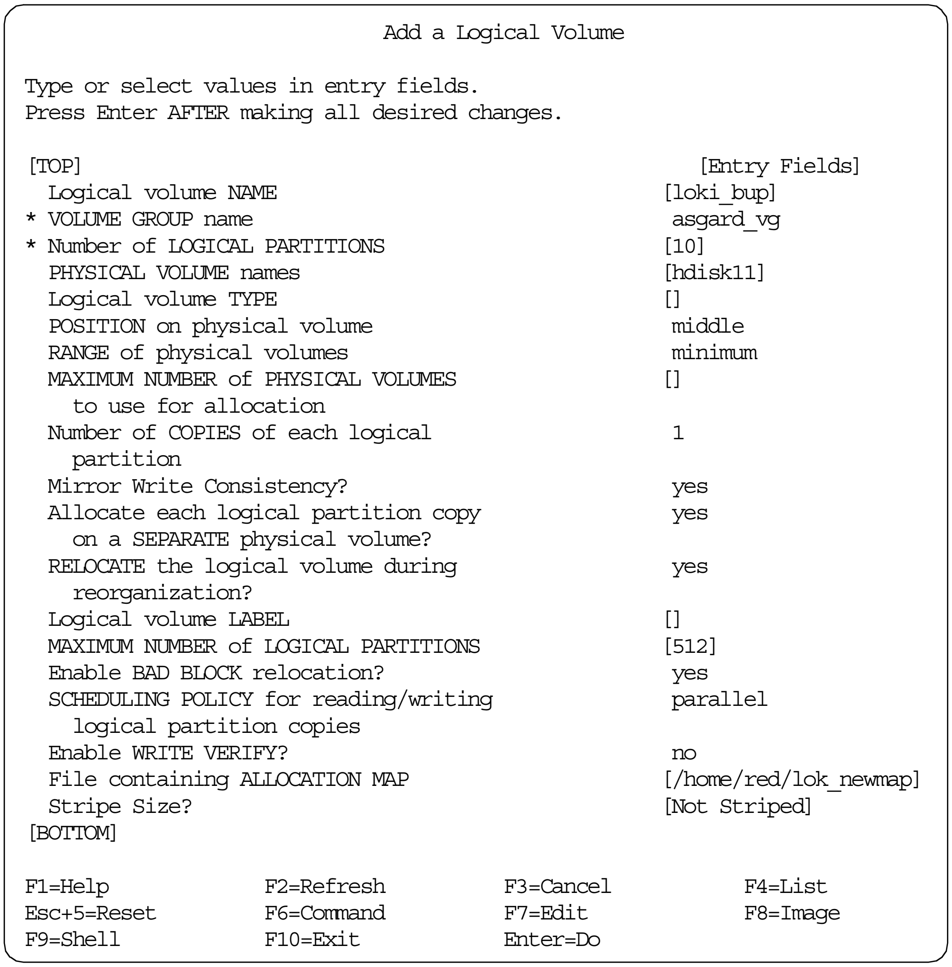
```
/home/red # lslv -m loki_lv
loki_lv:/home/loki
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0055 hdisk9            0055 hdisk10           0055 hdisk11
0002  0056 hdisk9            0056 hdisk10           0056 hdisk11
0003  0057 hdisk9            0057 hdisk10           0057 hdisk11
0004  0058 hdisk9            0058 hdisk10           0058 hdisk11
0005  0059 hdisk9            0059 hdisk10           0059 hdisk11
0006  0060 hdisk9            0060 hdisk10           0060 hdisk11
0007  0061 hdisk9            0061 hdisk10           0061 hdisk11
0008  0062 hdisk9            0062 hdisk10           0062 hdisk11
0009  0063 hdisk9            0063 hdisk10           0063 hdisk11
0010  0064 hdisk9            0064 hdisk10           0064 hdisk11
0011  0065 hdisk9            0065 hdisk10           0065 hdisk11
0012  0066 hdisk9            0066 hdisk10           0066 hdisk11
0013  0067 hdisk9            0067 hdisk10           0067 hdisk11
0014  0068 hdisk9            0068 hdisk10           0068 hdisk11
0015  0069 hdisk9            0069 hdisk10           0069 hdisk11
0016  0070 hdisk9            0070 hdisk10           0070 hdisk11
0017  0071 hdisk9            0071 hdisk10           0071 hdisk11
0018  0072 hdisk9            0072 hdisk10           0072 hdisk11
0019  0073 hdisk9            0073 hdisk10           0073 hdisk11
0020  0074 hdisk9            0074 hdisk10           0074 hdisk11
/home/red # lslv -m loglv00
loglv00:N/A
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0075 hdisk9            0075 hdisk10           0075 hdisk11
```

From this information, we have to create a map file; let us call it /home/red/ok_newmap. The second step is to remove one copy from loki_lv. This is done using smit.



```
                       Remove Copies from a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* LOGICAL VOLUME name                                  loki_lv
* NEW maximum number of logical partition              2
    copies
  PHYSICAL VOLUME names                               [hdisk11]







F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

We now create a new logical volume, loki_bup, reusing the physical partitions freed by the previous step. This is where the map file /home/red/ok_newmap becomes handy. We can physically match the new logical volume on top of the old copy of loki_lv.



```
                              Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
  Logical volume NAME                                 [loki_bup]
* VOLUME GROUP name                                    asgard_vg
* Number of LOGICAL PARTITIONS                        [10]
  PHYSICAL VOLUME names                               [hdisk11]
  Logical volume TYPE                                 []
  POSITION on physical volume                          middle
  RANGE of physical volumes                            minimum
  MAXIMUM NUMBER of PHYSICAL VOLUMES                  []
    to use for allocation
  Number of COPIES of each logical                     1
    partition
  Mirror Write Consistency?                            yes
  Allocate each logical partition copy                 yes
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                   yes
    reorganization?
  Logical volume LABEL                                []
  MAXIMUM NUMBER of LOGICAL PARTITIONS                [512]
  Enable BAD BLOCK relocation?                         yes
  SCHEDULING POLICY for reading/writing                parallel
    logical partition copies
  Enable WRITE VERIFY?                                 no
  File containing ALLOCATION MAP                      [/home/red/lok_newmap]
  Stripe Size?                                        [Not Striped]
[BOTTOM]

F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

We can check that the creation of our new logical volume with the exact fit worked, using the lslv command:

```
/home/red # lslv -m loki_bup
loki_bup:/home/backup
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0055 hdisk11
0002  0056 hdisk11
0003  0057 hdisk11
0004  0058 hdisk11
0005  0059 hdisk11
0006  0060 hdisk11
0007  0061 hdisk11
0008  0062 hdisk11
0009  0063 hdisk11
0010  0064 hdisk11
0011  0065 hdisk11
0012  0066 hdisk11
0013  0067 hdisk11
0014  0068 hdisk11
0015  0069 hdisk11
0016  0070 hdisk11
0017  0071 hdisk11
0018  0072 hdisk11
0019  0073 hdisk11
0020  0074 hdisk11
```

We now have to create a file system on top of our newly created logical file system. This is a trivial step:

```
/home/backup:
        dev             = /dev/loki_bup
        vfs             = jfs
        mount           = true
        options         = ro
        account         = false
```

The new file system is now mounted and we can backup the data and do whatever we want with this file system.

```
/home/backup # mount
  node       mounted        mounted over    vfs       date        options
-------- ---------------  ---------------  ------ ------------ ---------------
         /dev/hd4         /                jfs    Dec 31 18:53 rw,log=/dev/hd8
         /dev/hd2         /usr             jfs    Dec 31 18:53 rw,log=/dev/hd8
         /dev/hd9var      /var             jfs    Dec 31 18:53 rw,log=/dev/hd8
         /dev/hd3         /tmp             jfs    Dec 31 18:53 rw,log=/dev/hd8
         /dev/hd1         /home            jfs    Dec 31 18:54 rw,log=/dev/hd8
         /dev/lv00        /software        jfs    Dec 31 18:54 rw,log=/dev/lo
         /dev/stripelv    /stripefs        jfs    Jan 04 23:36 rw,log=/dev/st
         /dev/loki_lv     /home/loki       jfs    Jan 08 19:27 rw,log=/dev/lo
         /dev/loki_bup    /home/backup     jfs    Jan 08 23:43 ro
```

It is now time to merge the third copy of loki_lv again. This is done by using the exact same method. We remove the logical volume we created for the backup purpose and we recreate a new copy of loki_lv specifying the /home/red/ok_newmap map file.

```
/home/red # rmlv -f loki_bup
rmlv: Logical volume loki_bup is removed.


 mklvcopy -m'/home/red/lok_newmap' loki_lv 3 hdisk11
```

We now have to resynchronize the third copy of loki_lv with the `syncvg -l loki_lv` command and check that the physical partitions for the three copies are identical to what we had before this operation:

```
/home/red # lslv -m loki_lv
loki_lv:/home/loki
LP    PP1  PV1               PP2  PV2               PP3  PV3
0001  0055 hdisk9            0055 hdisk10           0055 hdisk11
0002  0056 hdisk9            0056 hdisk10           0056 hdisk11
0003  0057 hdisk9            0057 hdisk10           0057 hdisk11
0004  0058 hdisk9            0058 hdisk10           0058 hdisk11
0005  0059 hdisk9            0059 hdisk10           0059 hdisk11
0006  0060 hdisk9            0060 hdisk10           0060 hdisk11
0007  0061 hdisk9            0061 hdisk10           0061 hdisk11
0008  0062 hdisk9            0062 hdisk10           0062 hdisk11
0009  0063 hdisk9            0063 hdisk10           0063 hdisk11
0010  0064 hdisk9            0064 hdisk10           0064 hdisk11
0011  0065 hdisk9            0065 hdisk10           0065 hdisk11
0012  0066 hdisk9            0066 hdisk10           0066 hdisk11
0013  0067 hdisk9            0067 hdisk10           0067 hdisk11
0014  0068 hdisk9            0068 hdisk10           0068 hdisk11
```

6.3 Files and system backups

Once your system is in use, your next consideration should be backing up file systems, directories, and files. Files and directories represent a significant investment of time and effort. At the same time, all computer files are potentially easy to change or erase, either intentionally or by accident. If you take a careful and methodical approach to backing up your file systems, you should always be able to restore recent versions of files or file systems with little difficulty. When a hard disk crashes, the information contained on that disk is destroyed. The only way to recover the destroyed data is to retrieve the information from your backup copy.

For these reasons, the system administrator should develop and implement a backup strategy that meets the requirements of the particular site.

The following backup and restore procedures are also provided:

- Developing a backup strategy
- Compressing Files
- Backing up user files or file systems
- Backing up the system image and user-defined volume groups
- Backing up a user volume group
- Implementing scheduled backups
- Restoring from backup image individual user files

6.3.1 Backup methods

Several different methods exist for backing up information. One of the most frequently used methods is called backup by name, also called file name archive. This method of backup is done when the i flag is specified and is used to make a backup copy of individual files and directories. It is a method commonly used by individual users to back up their accounts.

Another frequently used method is called backup by file system, also called backup by i-node or file system archive. This method of backup is done when the i flag is not specified. It is used to make a backup copy of an entire file system and is the method commonly used by system administrators to back up large groups of files, such as all of the user accounts in /home. A file system backup allows incremental backups to be performed easily. An incremental backup will back up all files that have been modified since a specified previous backup.

The `compress` and `pack` commands enable you to compress files for storage, and the `uncompress` and `unpack` commands unpack the files once they have been restored. The process of packing and unpacking files takes time, but once packed, the data uses less space on the backup medium.

Several commands create backups and archives. Because of this, data that has been backed up needs to be labeled as to what command was used when doing the backup and how the backup was made (by name or by file system). The backup command is the most frequently used, but other commands serve specific purposes:

`backup` Backs up files by name or by file system.

`mksysb` Creates an installable image of the rootvg volume group.

`cpio` Copies files into and out of archive storage. Can usually read data archived on another platform provided it is in cpio format.

`dd` Converts and copies a file. Commonly used to convert and copy data to and from non-AIX systems, for example, mainframes. `dd` does not group multiple files into one archive; it is used to manipulate and move data.

`tar` Manipulates tar format archives.

`rdump` A network command that backs up files by file system onto a remote machine's device.

`pax` POSIX-compliant archive utility that can read and write tar and cpio archives.

6.3.2 Deciding on a backup policy

No single backup policy can meet the needs of all users. A policy that works well for a system with one user, for example, could be inadequate for a system that serves hundreds or thousands of users. Likewise, a policy developed for a system on which many files are changed daily would be inefficient for a system on which data changes infrequently. Whatever the appropriate backup strategy for your site, it is very important that one exist and that backups be done frequently and regularly. It is difficult to recover from data loss if a good backup strategy has not been implemented.

Only you can determine the best backup policy for your system, but the following points should be noted:

- Make sure that you can recover from major losses. Work through possible (albeit unlikely scenarios) and ensure that your strategy would work
- Check your backups periodically. Make sure that you can read the information on the backup media, it is also a good idea to check that the backup media is readable on another device.
- The cron facility can be used to schedule and verify backups out of hours. Different types of backup can also be scheduled for different times of the day/month.
- Keep old backups. Even if you use a regular cycle for re-using media, make sure that you keep some old copies, particularly for those occasions when the absence of an important file takes months to notice. It is also a good idea to have some backup media kept off site.
- Check what you are backing up. Not only check that you are making a backup of the correct information, but ensure it's integrity at the time. This also means checking that the files are not in use at the time.
- Fully back up systems prior to changes. This is rather self evident, and vitally important.

6.3.3 Restoring data

Once data has been properly backed up, there are several different methods of restoring the data based upon the type of backup command you used.

You need to know how your backup or archive was created to restore it properly. Each backup procedure gives information about restoring data. For example, if you use the `backup` command, you can specify a backup either by file system or by name. That backup must be restored the way it was done, by file system or by name.

Several commands restore backed up data, such as:

`restore`	Copies files created by the backup command.
`rrestore`	Network command that copies file systems backed up on a remote machine to the local machine.
`cpio`	Copies files into and out of archive storage.
`tar`	Manipulates archives. Used only for directories.

6.4 Backup commands

We will briefly examine some of the commands used to back up files and systems:

- `backup`
- `restore`
- `cpio`
- `tar`

6.4.1 The backup command

The `backup` command creates copies of your files on a backup medium, such as a magnetic tape or a diskette. Use the `backup` command when you want to back up large and multiple file systems. A level number can be specified to control how much data is backed up (full, 0; incremental, 1-9). Using the `backup` command is the only way you can specify the level number on backups.

The copies are in one of the two backup formats:

- Specific files backed up by name using the -i flag.
- Entire file system backed up by i-node using the Level and FileSystem parameters. One advantage of i-node backup is that the backup is defragmented when restored.

A single backup can span multiple volumes.

It should be noted that:

- Running the `backup` command results in the loss of all material previously stored on the selected output medium.

- Data integrity of the archive may be compromised if a file is modified during system backup. Keep system activity at a minimum during the system backup procedure.
- If a backup is made to a tape device with the device block size set to 0, it might be difficult to restore data from the tape unless the default write size was used with the `backup` command. The default write size for the backup command can be read by the `restore` command when the tape device block size is 0. In other words, the -b flag should not be specified when the tape device block size is 0. If the -b flag of the `backup` command is specified and is different from the default size, the same size must be specified with the -b flag of the `restore` command when the archived files are restored from the tape.

6.4.1.1 Backing up files by name

To back up by name, use the -i flag. The `backup` command reads standard input for the names of the files to be backed up. File types can be special files, regular files, or directories. When the file type is a directory, only the directory is backed up. The files under the directory are not backed up unless they are explicitly specified.

1. Files are restored using the same path names as the archived files. Therefore, to create a backup that can be restored from any path, use full path names for the files that you want to back up.
2. When backing up files that require multiple volumes, do not enter the list of file names from the keyboard. Instead, pipe or redirect the list from a file to the `backup` command. When you enter the file names from the keyboard and the backup process needs a new tape or diskette, the command "loses" any file names already entered but not yet backed up. To avoid this problem, enter each file name only after the archived message for the previous file has been displayed. The archived message consists of the character followed by the file name.
3. If you specify the -p flag, only files of less than 2 GB are packed.

6.4.1.2 Backing up file systems by i-node

To back up a file system by i-node, specify the Level and FileSystem parameters. When used in conjunction with the -u flag, the Level parameter provides a method of maintaining a hierarchy of incremental backups for each file system. Specify the -u flag and set the Level parameter to n to back up only those files that have been modified since the n-1 level backup. Information regarding the date, time, and level of each incremental backup is written to the /etc/dumpdates file. The possible backup levels are 0 to 9. A level 0 backup archives all files in the file system. If the /etc/dumpdates file contains no backup information for a particular file system, specifying any level causes all files in that file system to be archived.

The FileSystem parameter can specify either the physical device name (block or raw name) or the name of the directory on which the file system is mounted. The default file system is the root (/) file system.

Users must have read access to the file system device (such as /dev/hd4) or have backup authorization in order to perform backups by i-node.

You must first unmount a file system before backing it up by i-node. If you attempt to back up a mounted file system, a warning message is displayed. The `backup` command continues, but the created backup may contain inconsistencies because of changes that may have occurred in the file system during the backup operation.

Backing up file systems by i-node truncates the uid or gid of files having a uid or gid greater than 65535. When restored, these files may have different values for the uid and gid attributes. To retain the values correctly, always back up by name files having a uid or gid greater than 65535.

You can archive only JFS (journaled file system) file systems when backing up by i-node. Back up any non-JFS file systems by file name or by using other archive commands, such as `pax`, `tar`, or `cpio`.

It should be noted that backing up by i-node does not work properly for files that have UID (user ID) or GID (group ID) greater than 65535. These files are backed up with UID or GID truncated and will, therefore, have the wrong UID or GID attributes when restored. Backing up by name works properly for files that have a UID or GID greater than 65535.

As always, it is advisable to clean the media writing device (for example tape head) prior to writing the backup.

6.4.2 The restore command

The restore command reads archives created by the backup command and extracts the files stored on them. These archives can be in either file name or file system format. An archive can be stored on disk, diskette, or tape. Files must be restored using the same method by which they were archived. This requires that you know the format of the archive. The archive format can be determined by examining the archive volume header information that is displayed when using the -T flag. When using the -x, -r, -T, or -t flags, the restore command automatically determines the archive format.

Note

restore actively sparses files that are being restored. If a file has block aligned and sized areas that are NULL populated, then restoredoes not cause physical space for those filesystem blocks to be allocated. The size in bytes of the file remain the same, but the actual space taken within the filesystem is only for the non-NULL areas.

Individual files can be restored from either file name or file system archives by using the -x flag and specifying the file name. The file name must be specified as it exists on the archive. Files can be restored interactively from file system archives using the -i flag. The names of the files on an archive can be written to standard output using the -T flag.

Users must have write access to the file system device or have Restore authorization in order to extract the contents of the archive.

The diskette device, /dev/rfd0, is the default media for the restore command. To restore from standard input, specify a - (dash) with the -f flag. You can also specify a range of devices, such as /dev/rmt0-2.

Here are some of the main features of the restore commands:

- If you are restoring from a multiple-volume archive, the restore command reads the volume mounted, prompts you for the next volume, and waits for your response. After inserting the next volume, press **Enter** to continue restoring files.
- If an archive, created using the backup command, is made to a tape device with the device block size set to 0, it may be necessary for you to have explicit knowledge of the block size that was used when the tape was created in order to restore from the tape.
- Multiple archives can exist on a single tape. When restoring multiple archives from tape, the restore command expects the

input device to be a no-retension-on-open, no-rewind-on-close tape device. Do not use a no-rewind tape device for restoring unless either the -B, -s, or -X flag is specified. For more information on using tape devices, see the rmt special file.

6.4.2.1 File system archives

File system archives are also known as i-node archives due to the method used to archive the files. A file system name is specified with the `backup` command, and the files within that file system are archived based on their structure and layout within the file system. The `restore` command restores the files on a file system archive without any special understanding of the underlying structure of the file system.

When restoring file system archives, the `restore` command creates and uses a file named restoresymtable. This file is created in the current directory. The file is necessary for the `restore` command to do incremental file system restores.

6.4.2.2 File-Name Archives

File-name archives are created by specifying a list of file names to archive to the backup command. The restore command restores the files from a file-name archive without any special understanding of the underlying structure of the file system. The restore command allows for metacharacters to be used when specifying files for archive extraction. This provides the capability to extract files from an archive based on pattern matching. A pattern filename should be enclosed in single quotations, and patterns should be enclosed in brackets.

6.4.3 The cpio command

The `cpio` command is another command commonly used to copy files to backup media and back again. Like `tar`, `cpio` is not an AIX-specific command.

6.4.3.1 cpio -o Command

The `cpio -o` command reads file path names from standard input and copies these files to standard output, along with path names and status information. Avoid giving the `cpio` command path names made up of many uniquely linked files, as it may not have enough memory to keep track of them and would lose linking information.

6.4.3.2 cpio -i Command

The `cpio -i` command reads from standard input an archive file (created by the `cpio -o` command) and copies from it the files with names that match the Pattern parameter. These files are copied into the current directory tree.

You can list more than one Pattern parameter, using the file name notation described in the `ksh` command. Note that in this application the special characters * (asterisk), ? (question mark), and [...] (brackets and ellipses) match the / (slash) in path names, in addition to their use as described in the `ksh` command. By default, all files from the input archive file are selected.

It is important to redirect the output from the `cpio` command to a raw file (device) and not the block device. This is because writing to a block device is done asynchronously and there is no way to know if the end of the device is reached.

Note

The `cpio` command is not enabled for files greater than 2 GB in size due to limitations imposed by XPG/4 and POSIX.2 standards.

`cpio` does not preserve the sparse nature of any file that is sparsely allocated. Any file that was originally sparse before the restoration will have all space allocated within the filesystem for the size of the file.

6.4.4 The tar command

The `tar` command is also used to write and retrieve files from an archive file or device.

When writing to an archive, the `tar` command uses a temporary file (the /tmp/tar* file) and maintains in memory a table of files with several links. You receive an error message if the tar command cannot create the temporary file or if there is not enough memory available to hold the link tables.

The `tar` command manipulates archives by writing files to, or retrieving files from an archive storage medium. The files used by the `tar` command are represented by the File parameter. If the File parameter refers to a directory, then that directory and recursively all files and directories within it are referenced as well.

Due to limitations on header block space in the tar command, user numbers (UIDs) and group identification numbers (GIDs) larger than 65,535 will be corrupted when restored to certain systems. The size constraint affects only the ownership and permissions, causing no damage to the data.

> **Note**
>
> The `tar` command is not enabled for files greater than 2 GB in size due to limitations imposed by XPG/4 and POSIX.2 standards.
>
> `tar` does not preserve the sparse nature of any file that is sparsely allocated. Any file that was originally sparse before the restoration will have all space allocated within the file system for the size of the file.
>
> `tar` has no recovery from tape errors

6.4.5 Block sizes and performance

When data is written to tape it is written in blocks. The blocks on a tape are separated by inter-record gaps. It is important to understand the structure of the written tape in order to understand the problems that can occur with changing block sizes.

In fixed block-size mode all blocks on the tape are the same size. They are the size of the block size set in the device configuration. All read()s and write()s to the tape drive must be a multiple of the fixed block size. Also aread() will return as many blocks as needed to satisfy the read() request. If a file mark is encountered during the read, only the data up to the file mark will be returned. It is not possible for the tape drive to read a tape whose block size is not the same as the block size in the device configuration.

In variable block-size (0) mode, the blocks written on the tape are the size of the read() and write() requests to the device driver. In this case, the actual block sizes on the tape can be changed using options of the backup commands:

- `tar -b`
- `cpio -C`
- `backup -b`

In variable mode, read() requests greater than the size of the block on the tape will return only the data from the next block on the tape. It is this feature that allows tapes written in any block size (fixed or variable) to read with the `dd` command (the output from the `dd` command may be piped to restore, `tar`, or `cpio`, for example.)

Note

The backup, tar, and cpio commands cannot read all tapes by using a large block size because they assume there is an error if they get a short read().

There are some recommendations for particular tape drives:

8 mm tape drive The use of a fixed block size that is not a multiple of 1 K is inefficient. The 8 mm tape drive always writes internally in 1 K blocks. It simulates the effect of variable block sizes. For example, using a fixed block size of 512 bytes (or using variable block size and writing 512 bytes at a time) wastes one half of the tape capacity and decreases the maximum transfer rate.

Magstar tape drive It is recommended that the block size be set to 262144 or 0. Otherwise, the commands can be used to set the block size:

- mksysb -b512
- sysback -b256
- savevg -b512
- backup -b512
- tar -N512
- dd isb=512, obs=512, or bs=512
- cpio -C512

-IBM 9348 Magnetic Tape Unit Model 12:

- Default block size of 32 k

6.5 Backing up and restoring volume groups

AIX provides two commands that will backup and restore volume groups. Thus, a whole volume group can be recovered without the administrator needing to understand the principles of creating volume groups, logical volumes and file systems.

The commands are:

- savevg

- restvg

6.5.1 The savevg command

The savevg command finds and backs up all files belonging to a specified volume group. A volume group must be varied-on, and the file systems must be mounted. The savevg command uses the data file created by the mkvgdata command. This file can be one of the following:

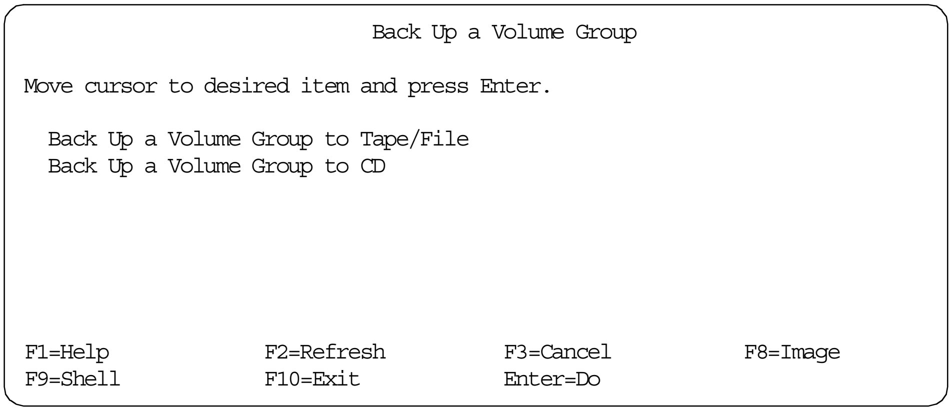
/image.data
Contains information about the root volume group (rootvg). The savevg command uses this file to create a backup image that can be used by Network Installation Management (NIM) to reinstall the volume group to the current system or to a new system.

/tmp/vgdata/vgname/vgname.data
Contains information about a user volume group. The vgname variable reflects the name of the volume group. The savevg command uses this file to create a backup image that can be used by the restvg command to remake the user volume group.

> **Note**
>
> The savevg command will not generate a bootable tape if the volume group is not the root volume group.

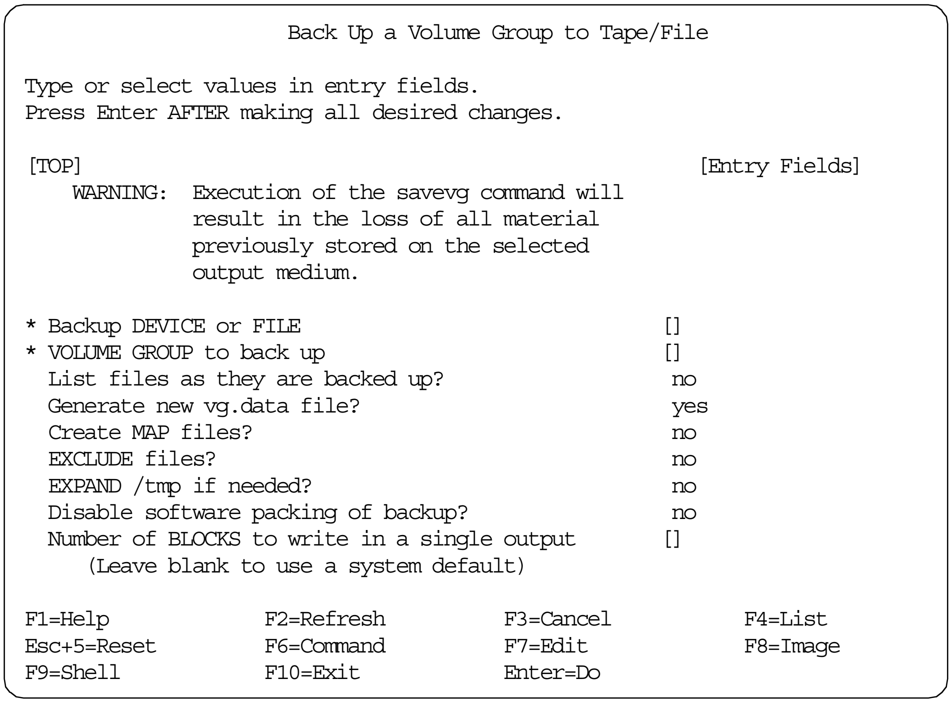
The savevg command can be run either from the command line or through the smit vgbackup menu shown in the following screen.

```
smit vgbackup
```



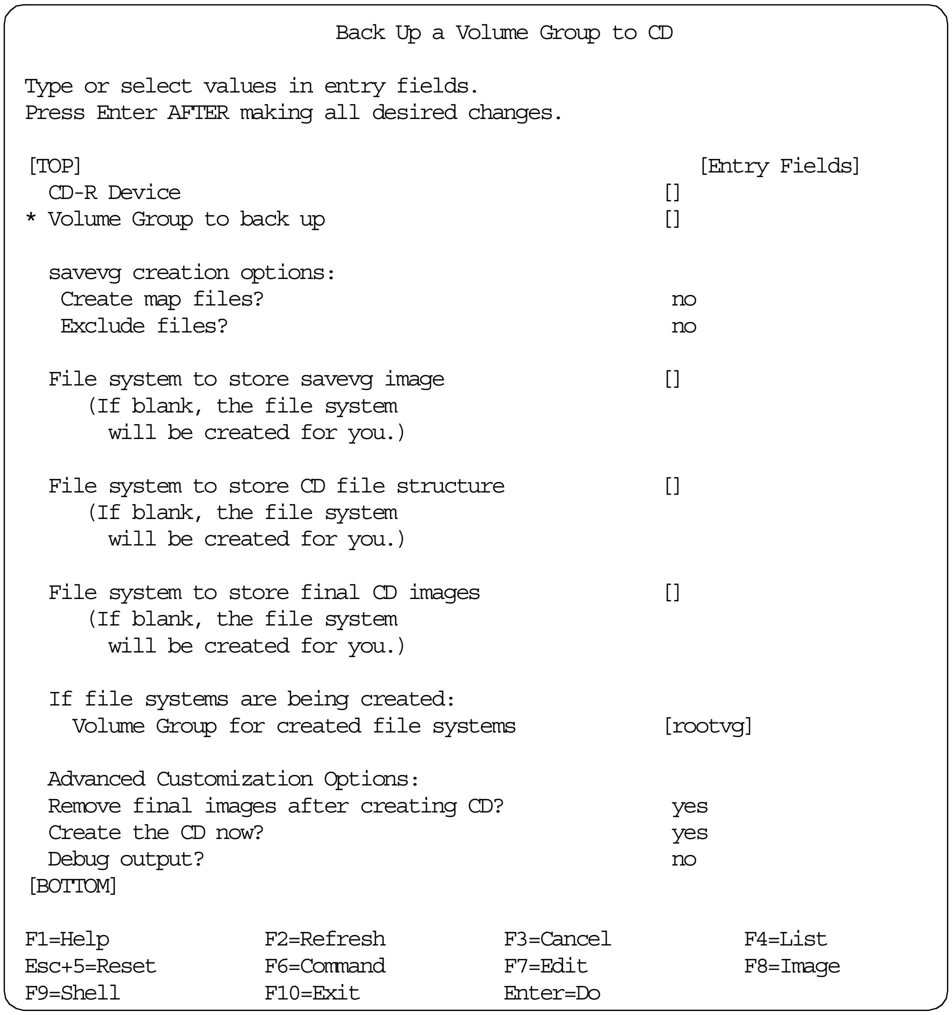
```
                              Back Up a Volume Group

Move cursor to desired item and press Enter.

  Back Up a Volume Group to Tape/File
  Back Up a Volume Group to CD








F1=Help             F2=Refresh          F3=Cancel           F8=Image
F9=Shell            F10=Exit            Enter=Do
```

The smit savevg options are shown below.



```
                      Back Up a Volume Group to Tape/File

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                   [Entry Fields]
    WARNING:  Execution of the savevg command will
              result in the loss of all material
              previously stored on the selected
              output medium.

* Backup DEVICE or FILE                              []
* VOLUME GROUP to back up                            []
  List files as they are backed up?                   no
  Generate new vg.data file?                          yes
  Create MAP files?                                   no
  EXCLUDE files?                                      no
  EXPAND /tmp if needed?                              no
  Disable software packing of backup?                 no
  Number of BLOCKS to write in a single output       []
     (Leave blank to use a system default)

F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

A new command has been added that will allow a user to copy an existing savevg image or create a new one onto a recordable CD-ROM. Here is the smit menu associated with the backup of a volume group on a CDR.



```
                          Back Up a Volume Group to CD

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                     [Entry Fields]
  CD-R Device                                          []
* Volume Group to back up                              []

  savevg creation options:
   Create map files?                                    no
   Exclude files?                                       no

  File system to store savevg image                    []
     (If blank, the file system
       will be created for you.)

  File system to store CD file structure               []
     (If blank, the file system
       will be created for you.)

  File system to store final CD images                 []
     (If blank, the file system
       will be created for you.)

  If file systems are being created:
    Volume Group for created file systems              [rootvg]

  Advanced Customization Options:
  Remove final images after creating CD?                yes
  Create the CD now?                                    yes
  Debug output?                                         no
[BOTTOM]

F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

6.5.2 The restvg command

The `restvg` command restores the user volume group and all its containers and files, as specified in the /tmp/vgdata/vgname/vgname.data file (where vgname is the name of the volume group) contained within the backup image created by the `savevg` command.

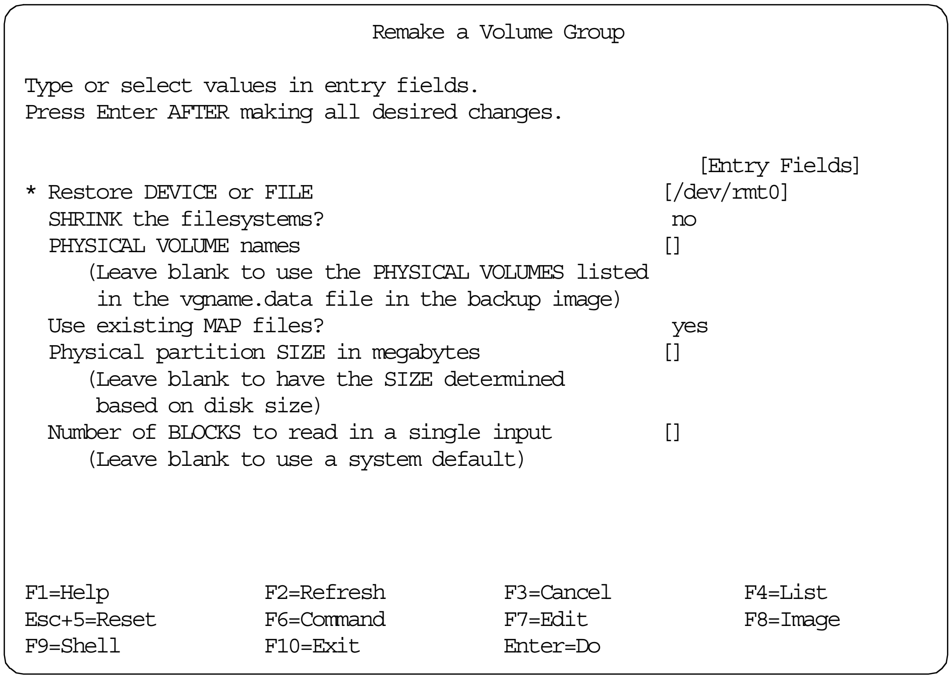
The `restvg` command restores a user volume group. The `bosinstall` routine reinstalls the root volume group (rootvg). If the `restvg` command encounters a rootvg volume group in the backup image, the `restvg` command exits with an error.

If a yes value has been specified in the EXACT_FIT field of the logical_volume_policy stanza of the /tmp/vgdata/vgname/vgname.data file, the `restvg` command uses the map files to preserve the placement of the physical partitions for each logical volume. The target disks must be of the same size or larger than the source disks specified in the source_disk_data stanzas of the vgname.data file.

> **Note**
>
> To view the files in the backup image or to restore individual files from the backup image, the user must use the `restore` command with the -T or -x flag, respectively. (Refer to the `restore` command for more information.)

The `restvg` command can be executed from the command line or through smit. This particular option will recreate the volume group, logical volumes and file systems. Then restore all the files.

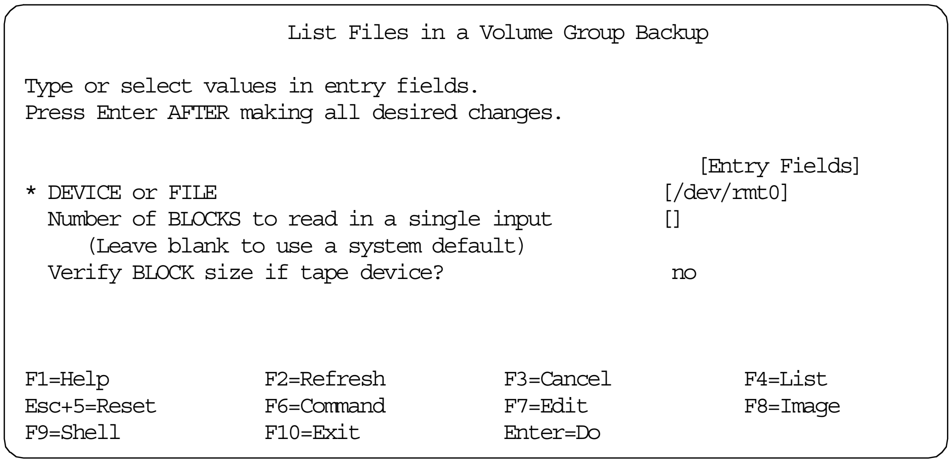
```
smit restvg
```



```
                              Remake a Volume Group

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                         [Entry Fields]
* Restore DEVICE or FILE                              [/dev/rmt0]
  SHRINK the filesystems?                              no
  PHYSICAL VOLUME names                               []
     (Leave blank to use the PHYSICAL VOLUMES listed
      in the vgname.data file in the backup image)
  Use existing MAP files?                              yes
  Physical partition SIZE in megabytes                []
     (Leave blank to have the SIZE determined
      based on disk size)
  Number of BLOCKS to read in a single input          []
     (Leave blank to use a system default)




F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

There is also a smit menu to list the files on a file system backup image.

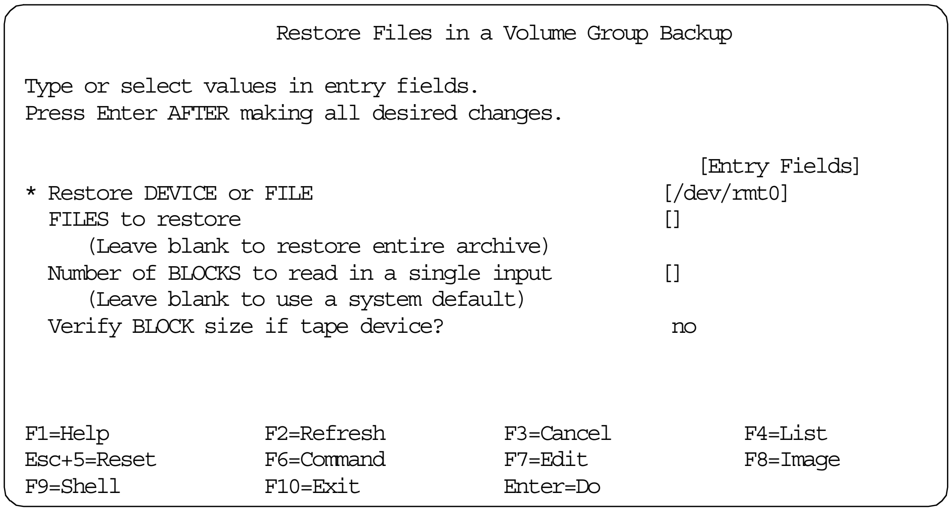
```
smit lsbackvg
```



```
                      List Files in a Volume Group Backup

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
* DEVICE or FILE                                       [/dev/rmt0]
  Number of BLOCKS to read in a single input           []
     (Leave blank to use a system default)
  Verify BLOCK size if tape device?                     no



F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

Individual files can also be restored from a `savevg` backup.

```
smit restsavevg
```



```
                     Restore Files in a Volume Group Backup

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                                          [Entry Fields]
* Restore DEVICE or FILE                               [/dev/rmt0]
  FILES to restore                                     []
     (Leave blank to restore entire archive)
  Number of BLOCKS to read in a single input           []
     (Leave blank to use a system default)
  Verify BLOCK size if tape device?                     no



F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         F6=Command          F7=Edit             F8=Image
F9=Shell            F10=Exit            Enter=Do
```

6.6 The mksysb command

The `mksysb` command creates a backup of the operating system (that is, the root volume group). You can use this backup to reinstall a system to its original state after it has been corrupted. If you create the backup on tape, the tape is bootable and includes the installation programs needed to install from the backup.

The file system image is in backup-file format. The tape format includes a boot image, a bosinstall image, and an empty table of contents followed by the system backup (root volume group) image. The root volume group image is in backup-file format (it used to be in tar format in an earlier release of AIX, Version 3.2), starting with the data files and then any optional map files.

The `mksysb` command also creates the /bosinst.data file from the system default file if the /bosinst.data file does not exist.

The `mksysb` command uses a file (/image.data) to rebuild the rootvg and associated logical volumes and file systems. This file is created by the `mkszfile` command. In AIX Version 3.x, this file was called /.fs.size.

It should be noted that:

- The image the `mksysb` command creates does not include data on raw devices or in user-defined paging spaces.
- If you are using a system with a remote-mounted /usr file system, you cannot reinstall your system from a backup image.
- The `mksysb` command may not restore all device configurations for special features, such as /dev/netbios and some device drivers not shipped with the product.
- Some rspc systems do not support booting from tape. When you make a bootable mksysb image on an rspc system that does not support booting from tape, the `mksysb` command issues a warning indicating that the tape will not be bootable. You can install a mksysb image from a system that does not support booting from tape by booting from a CD and entering maintenance mode. In maintenance mode you will be able to install the system backup from tape.

The smit menus allow users to:

- Create a mksysb backup
- List the files
- Restore particular files

The root volume group can be rebuilt by booting either from the mksysb tape, or installing media at the same level as the mksysb image.

With the latest version of AIX, 4.3.3, the `mksysb` command also support a Recordable CD-ROM as a output media.

6.6.1 Smit fast path for the mksysb related menus

smit includes several menus associated with backing up the rootvg volume group. Here is a list of the fast path associated with those menus:

backsys	Choice between back up the system, list files in a system backup, or restore files from the backup image.
sysbackup	Choice between back up the system to a tape or a file, back up the system on a CD, or create a generic backup CD.
mksysb	Back up the system on a tape or a file.
mkcd	Back up the system on a recordable CD-ROM.
lsmksysb	List the files included in a system image.
restmksysb	Restore files from a system image.

Chapter 7. Performance

All kinds of applications, such as a Journaled File System (JFS) or a Relational DataBase Management System (RDBMS), are using logical volumes for storing their data. These applications often have a need for fast disk access. But how can you achieve this? How can you achieve the best LVM performance?

This chapter discusses the performance of your system from the Logical Volume Manager perspective. If you suspect that the LVM configuration is not optimal, this is the chapter to read. You will find here advice on how to configure the logical volumes to achieve the best performance.

7.1 Introduction

By far the slowest operation for a running program (other than waiting on human interaction) is reading from, or writing data to, a disk. Several factors play a role in this:

- The disk controller must be directed to access the specified blocks (queuing delay).
- The disk arm must seek to the correct cylinder (seek latency).
- The read/write heads must wait until the correct block rotates under them (rotational latency).
- The data must be transmitted to the controller (transmission time) and then conveyed to the application program (interrupt handling time).
- Disk operations can have many causes besides explicit read or write requests in the program. System tuning activities frequently turn out to be hunts for unnecessary disk I/O.

When a user application is installed on your system, it generates read and write actions to fixed disk drives. When read and write actions occur, they trigger disk head movements. The goal is to make sure reading and writing is done efficiently; thus, when trying to reduce the number of disk head movements, make sure that seek time is as minimal as possible. In general, applications do I/O operations on a regular basis. If your I/O subsystem is not fast enough due to a bad configuration or slow disks, the application will not perform the way you want it to. The I/O operations will be put in an I/O queue and wait until their turn to be processed has come. Users will have to wait until the I/O request they submitted is handled by the I/O subsystem. And, in general, users do not like to wait. Of course, performance is not only dependent on the I/O subsystem. If you are tuning your entire system, you should consider the rules mentioned in the *AIX Performance and Tuning Guide*, SC23-2365. As a reminder, the tuning process is shown in Figure 83.

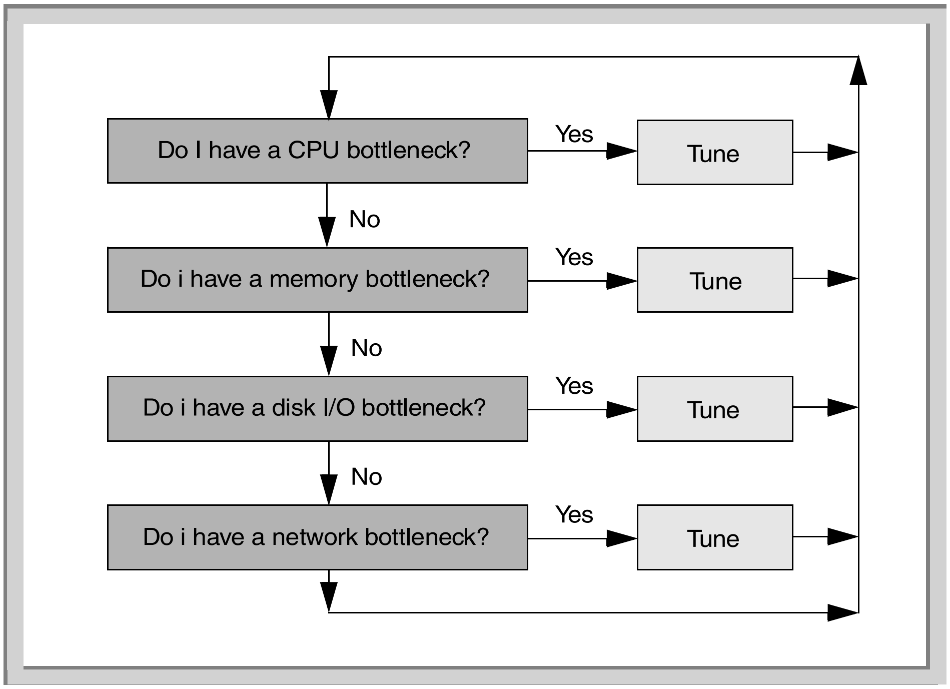

Do I have a CPU bottleneck?
Yes
Tune
No
Do i have a memory bottleneck?
Yes
Tune
No
Do i have a disk I/O bottleneck?
Yes
Tune
No
Do i have a network bottleneck?
Yes
Tune

Figure 83. The tuning process

Remember that the tuning process is an ongoing process. If you change something, you have to start again at the beginning of the tuning process.

This chapter only covers tuning on the third level, which is the I/O section on the logical volume and the journaled file system level. The issues we cover in this chapter are mentioned below. We discuss logical volume manager performance issues starting with the physical layer as shown in Figure 84.

- The physical layer
- The logical layer
  - Mirroring
  - Striping
- The application layer
- Reorganizing elements

7.2 The physical layer

In the first chapter, we talked about the logical volume manager as a layered structure. The first layer within that structure is the physical layer that is made out of cabling, disk adapters, system busses, fixed-disk drives, and the AIX device drivers. Because this redbook does not focus on performance in general, detailed information about these components is beyond the scope of this book. For detailed performance issues on the physical level, please read the *AIX Performance and Tuning Guide*, SC23-2365. Some basic tips, however, will be mentioned.

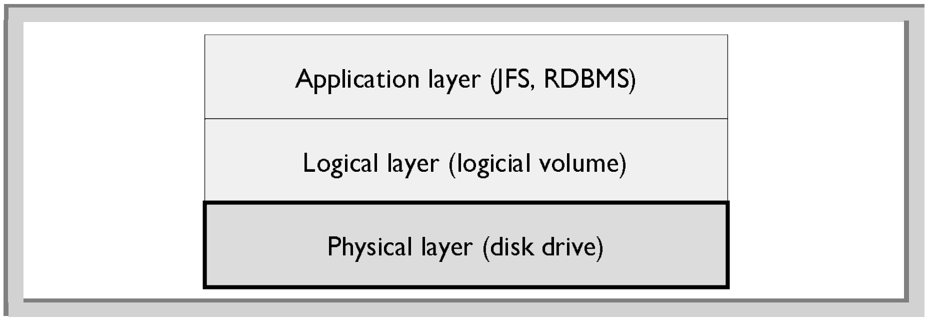

Application layer (JFS, RDBMS)
Logical layer (logicial volume)
Physical layer (disk drive)

Figure 84. LVM components, the physical layer

When adding disk drives to your system, these are some things you should keep in mind:

- The placement of adapters
- The number of disk adapters used
- The configuration of the adapters
- The type of disks used
- The cabling configuration

7.2.1 Adapters

At this time, several high-speed adapters for connecting disk drives are available; however, if you build a configuration with a lot of these adapters on one system bus, you may have fast adapters, but the bus becomes the performance bottleneck. Therefore, it is always better to spread these fast adapters across several busses. When you use PCI adapters, make sure you check the *PCI Adapter Placement Guide*, SA23-2504, for the proper adapter locations.

When you use the SSA adapters, you should look at the size of the read/write operations. An SSA Enhanced Raid adapter (feature code #-6215) can handle as many as 2200 - 3100 I/Os per second (30 - 40 MBps). An IBM Advanced Serial RAID SSA adapter (feature code #6225) can handle as many as 7000 I/Os per second (90 MBps). Read or write operation with a relative small I/O buffer size leads to a lower throughput than one with a larger buffer size. This is due to both the CPU and the adapter congestion.

7.2.2 The number of disk adapters used

Attaching forty fast disks to one disk adapter may result in low disk performance. Although the disks are fast, all the data must go through that one adapter. This adapter gets overloaded, resulting in low performance. Adding more disk adapters is a good thing to do when a lot of disks are used, especially if those disks are used intensively.

7.2.3 The configuration of the adapter

Just adding disks to several adapters across several busses does not give you maximum performance in every case. For SCSI peripherals, several adjustments can be made to improve performance. AIX has the ability to enforce limits on the number of I/O requests that can be outstanding from the SCSI adapter to a given SCSI bus or disk drive. Detailed information about these setting are beyond the scope of this book and can be found in the *AIX Performance and Tuning Guide*, SC23-2365.

7.2.4 Type of disks used

Several physical disks are available, and all have different performance specifications. Older disks can transfer up to 7 MBps while newer disks can transport up to 20 MBps. Mixing disks with variable performance specifications may lead to confusion when measuring performance.

7.2.5 The cabling configuration

Cabling is very important, especially in SSA configurations. An SSA adapter can have more than one loop, and dividing the fixed-disk drives across those loops will give you more bandwidth. Figure 85 shows you a configuration that is not recommended because the disk drives are not spread across the SSA adapter.

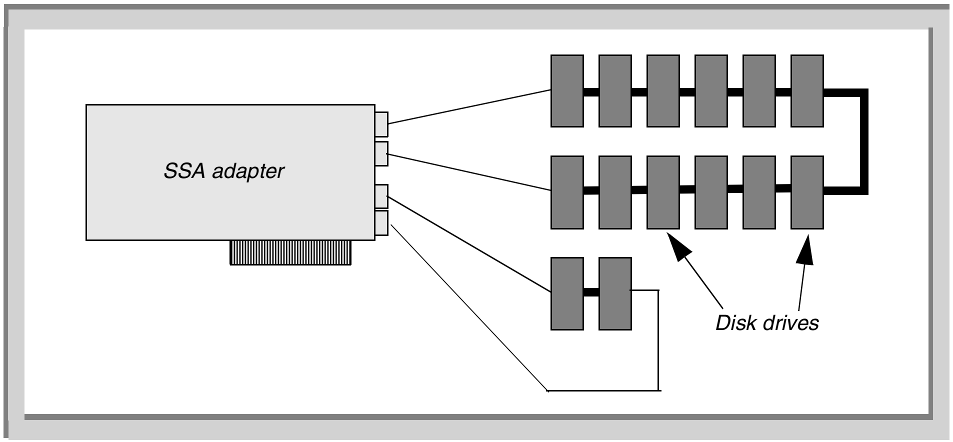


Figure 85. Non-recommended SSA cabling configuration

Figure 86 shows a better way of spreading the SSA disk drives across the two SSA ports.

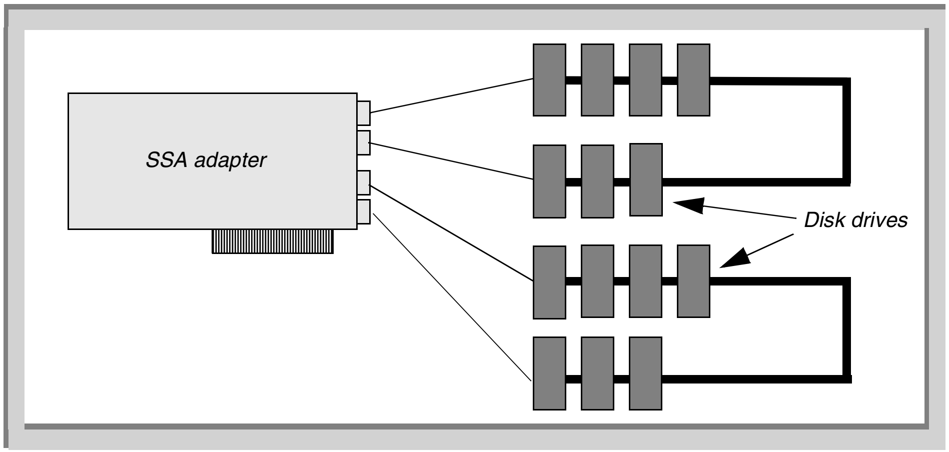


Figure 86. Better SSA cabling configuration

7.3 The logical layer

The logical layer, as shown in Figure 87, is also referred to as the Logical Volume Manager (LVM). Besides volume groups and logical volumes, the software handling mirroring and striping is also part of this layer.

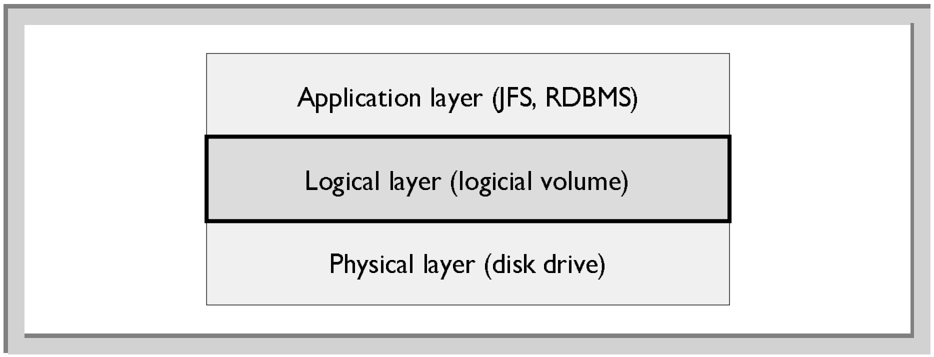

Application layer (JFS, RDBMS)
Logical layer (logicial volume)
Physical layer (disk drive)

Figure 87. LVM components, the logical layer

7.3.1 Volume groups

When you want to store data on a physical disk, you will need to add this physical disk into a volume group. Once this is done, you can create a logical volume on the physical disk. Only logical volumes can be used for storing data. There are no performance related options when creating a volume group with `mkvg`

Avoid adding disks with various performance characteristics into a single volume group, because the logical volume manager does not distinguish what type of physical disk you add to a volume group, you are allowed to add a slow SCSI-1 disks to a volume group together with faster SSA physical disk drives. Reading or writing a lot of data to such a volume group may be slowed down if you use a striped or mirrored configuration.

7.3.2 Logical volumes

Creating a logical volume can be done in several ways. However if you want the best performance, you should consider the following performance issues before creating them.

7.3.2.1 Logical volume placement

When you create a logical volume, the application on top of it will use this logical volume to store data. This means read and write actions will occur which implicates disk head movements. Fixed-disk drives are able to transport data faster and faster these days but are still slow compared to memory. Therefore, make sure reading and writing is done efficientlty, thus trying to reduce the number of disk head movements. Storing data contiguous on the disk platter is generally a good start.

7.3.2.2 Creating logical volumes

When you create a logical volume using the mklv command, there are several options that affect performance. One of them is the placement of the logical volume. Five locations are available on the physical disk. We will explain the implacation of choosing one over another later. Another way to minimize disk overloading is to spread a logical volume across several physical disks. You may also want to configure your logical volumes on exact disk locations. For this, you need to use map files. To make sure that the reorgvg command does not change the placement of the logical volumes, you could prevent relocation of the logical volume. All these options can be passed to the mklv command. Here is the syntax of the mklv command:

```
mklv [-a IntraPolicy] [-b BadBlocks] [-c Copies] [-d Schedule]
     [-e InterPolicy] [-i] [-L Label] [-m MapFile] [-r Relocate]
     [-s Strict] [-t Type] [-u UpperBound] [-v Verify] [-w MWC]
     [-x MaxLPs] [-y LVname][-Y Prefix] [-S StripeSize] [-U userid]
     [-G groupid] [-P modes] VGname NumberOfLPs [PVname...]
```

The options of mklv that are relevant from a performance point of view are:

```
-a  Position
```

If you create a logical volume either using SMIT or the command line interface with its default parameters, the -a flag is not passed to the mklv command. What happens is that this logical volume is placed in the outer middle location as shown in Figure 88. The command used to create the logical volume is:

```
brenzef[root] # mklv -y ronaldlv rootvg 10
```

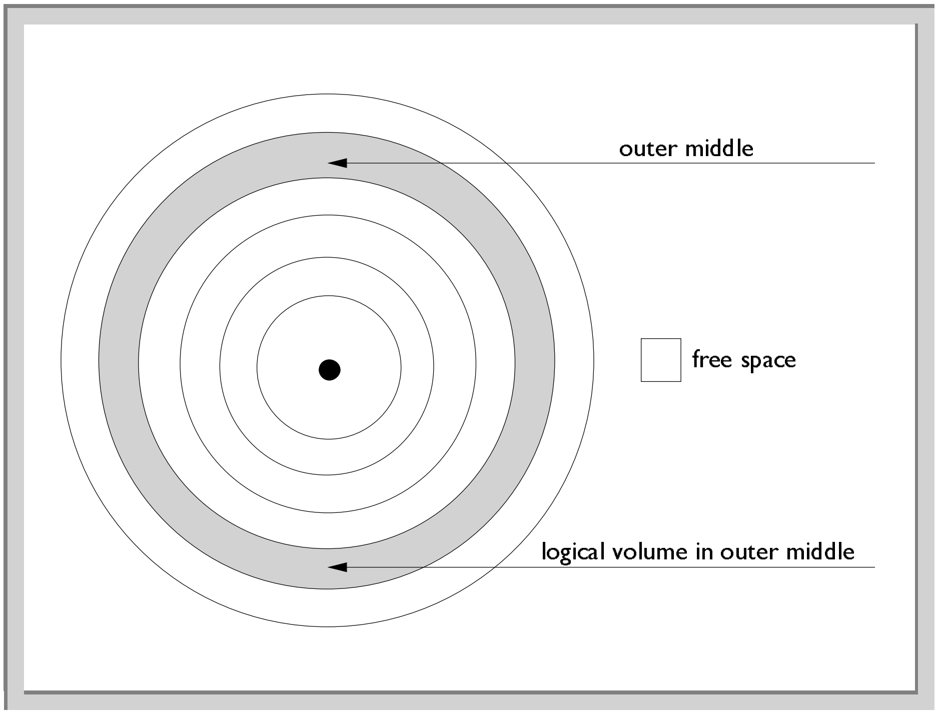


Figure 88. A disk with one logical volume

If you create just one logical volume on the disk, this should not be a problem, but it would be a waste of disk space since you probably want to create more logical volumes on that drive. If you create another logical volume without supplying the `mklv` command with the appropriate `-a` location option, again `mklv` creates it in the outer middle location. But, since you already created a logical volume in this location, no physical partitions are available to fulfill allocation anymore. So, what `mklv` does is allocate the physical partitions close to the desired location as shown in Figure 89 on page 311.

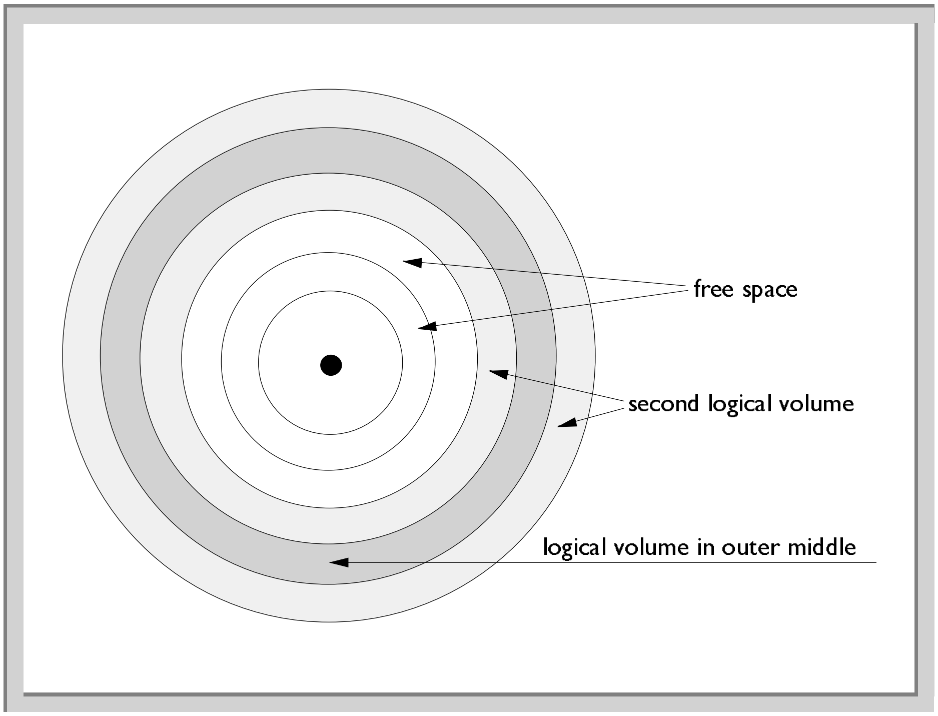


Figure 89. Fragmented disk distribution

You can see in Figure 89 that the second logical volume is fragmented. This can cause extra disk head movements when a process does a read or write operation on the edges of both the outer edge and center locations. A better way to create a logical volume is to instruct the `mklv` command by using the `-a Position` flag to select another region. If you pass the `-a im` option to `mklv`, the logical volume will be created in the inner-middle location as shown in Figure 90. An example on how to create a logical volume on the inner middle location would be:

```
brenzef[root] # mklv -y ronaldlv -a im rootvg 10
```

Finding fragmented logical volumes is described in Section 7.8, “Finding fragmented logical volumes” on page 329.

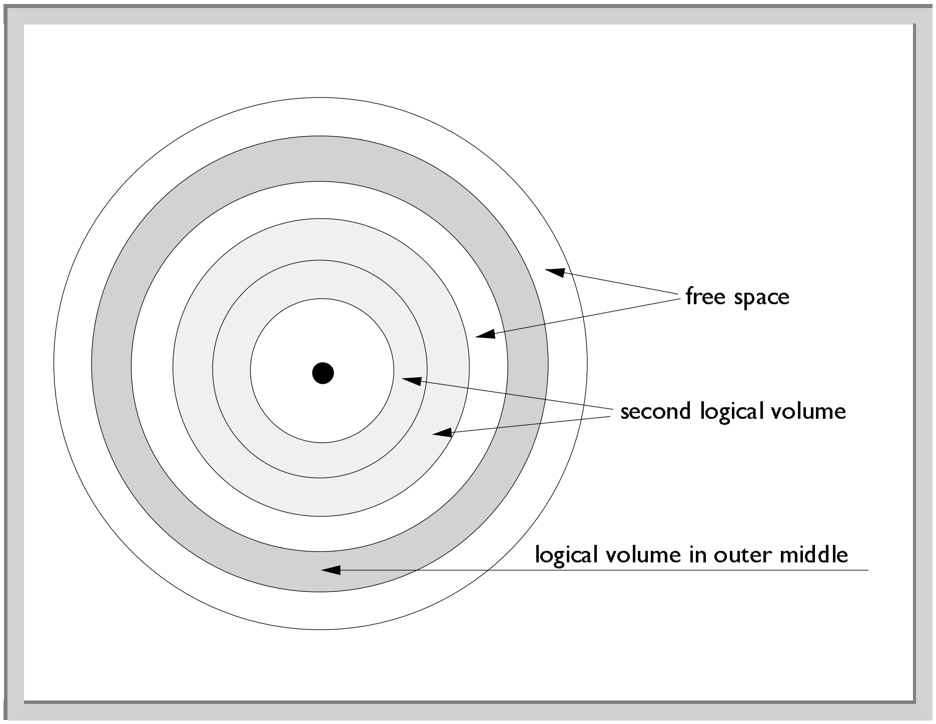


Figure 90. Disk distribution with mklv location flag used

The `mklv -a` options for selecting these regions are:

`im` Allocates logical partitions in the inner-middle section of each physical volume.

`ie` Allocates logical partitions in the inner-edge section of each physical volume.

`c` Allocates logical partitions in the center section of each physical volume.

`m` Allocates logical partitions in the outer-middle section of each physical volume. (This is the default position.)

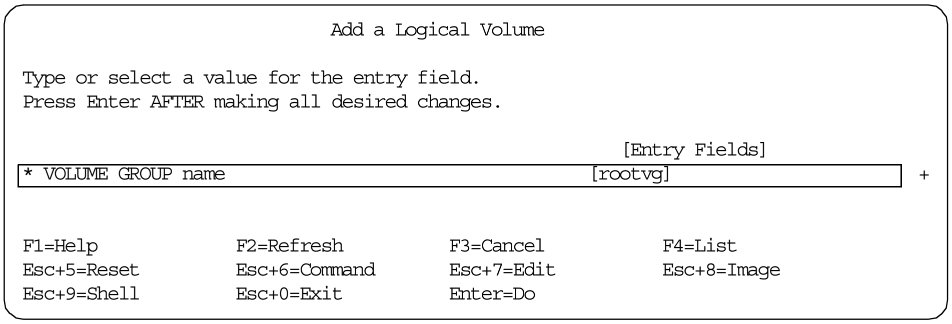
`e` Allocates logical partitions in the outer edge section of each physical volume.

Selecting another region can be done by either using the command line or by using the popup list of SMIT. If you want to use SMIT, you should enter the following:

```
brenzef[root] # smit mklv
```

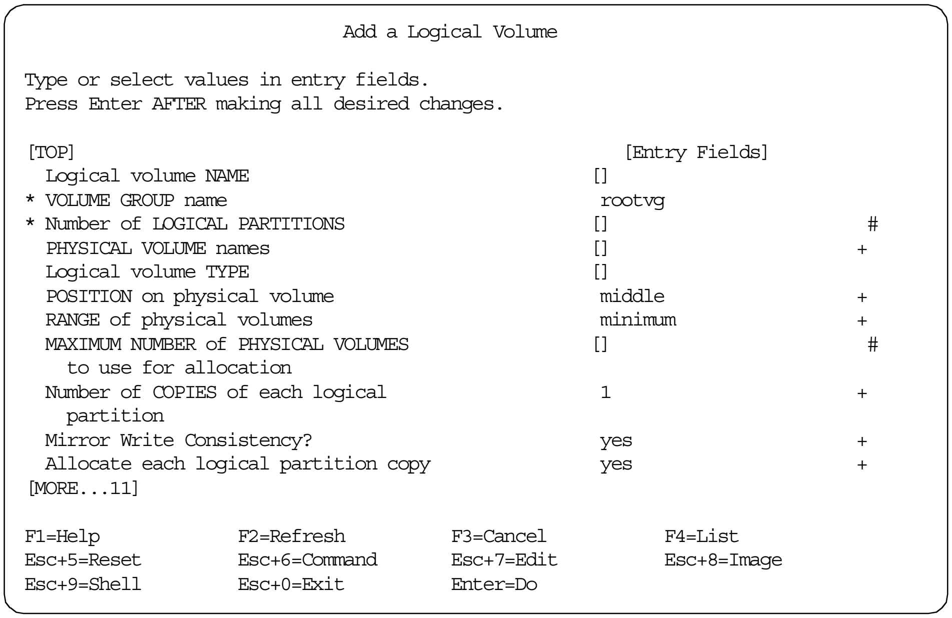
Select **Add A Logical Volume**.

Enter the volume group name.



```
                              Add a Logical Volume

Type or select a value for the entry field.
Press Enter AFTER making all desired changes.

                                                     [Entry Fields]
* VOLUME GROUP name                                  [rootvg]          +


F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         Esc+6=Command       Esc+7=Edit          Esc+8=Image
Esc+9=Shell         Esc+0=Exit          Enter=Do
```

All `mklv` options can be set by using the SMIT menu as shown in the following screen.



```
                               Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[TOP]                                                  [Entry Fields]
  Logical volume NAME                               []
* VOLUME GROUP name                                  rootvg
* Number of LOGICAL PARTITIONS                      []                        #
  PHYSICAL VOLUME names                             []                       +
  Logical volume TYPE                               []
  POSITION on physical volume                        middle                  +
  RANGE of physical volumes                          minimum                 +
  MAXIMUM NUMBER of PHYSICAL VOLUMES                []                        #
    to use for allocation
  Number of COPIES of each logical                   1                       +
    partition
  Mirror Write Consistency?                          yes                     +
  Allocate each logical partition copy               yes                     +
[MORE...11]

F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         Esc+6=Command       Esc+7=Edit          Esc+8=Image
Esc+9=Shell         Esc+0=Exit          Enter=Do
```

7.3.3 Where should you create logical volumes?

Where to create a logical volume depends on the application if it uses the logical volume for sequential or random access, and if the application does many reads or writes.

- Sequential writing

A logical volume used for sequential writing should be located on the outer edge because that is where the track's density is higher. In a mirrored configuration the logical volume should be close to the MWCC (Mirror Write Consistency Check region), that makes this location most suitable for these kind of operations.

- Random writing

  If you do random writes the outer edge is the best location. The tracks density on the outer edge is higher, so more data is located in that region. If the space in the outer edge is not enough for the logical volume and it has to be extended to outer middle or middle the performance, the gain is reduced to zero. If you consider a mirrored configuration. These random writes will probably be in different logical track groups, so in a mirrored configuration the MWCC is updated more frequently. Creating a logical volume close to the MWCC location will reduce disk head movements. Creating a mirrored random written logical volume on the inner edge gives you a substantial performance degradation.

- Random/Sequential reads

  For random and sequential reads the center location is the best location. The head is statistical the quickest in the center location, that makes this location favorable compared to the other regions.

These recommendations are fine when you have an application that does only lots of sequential/random reads and writes, but if I have an application that does both? sometimes sequential and sometimes random?

The logical volumes for applications that do a mixture of sequential/random read/write should be placed on the outer edge location. If the majority is random write place this logical volume on the edge. If the majority is sequential read place them in the outer middle or on the center.

Logical volumes that are only read from can be placed on the inner edge and inner middle.

Note

Do not place write mirrored sequential accessed logical volume on the inner edge.

7.4 Fragmentation

When you create a logical volume it should be contiguous. And if you create a logical volume on an unused disk, the logical volume will be contiguous in most cases.

7.4.1 Fragmentation example 1

Imagine the following situation where we create a logical volume and do some extension and removal.

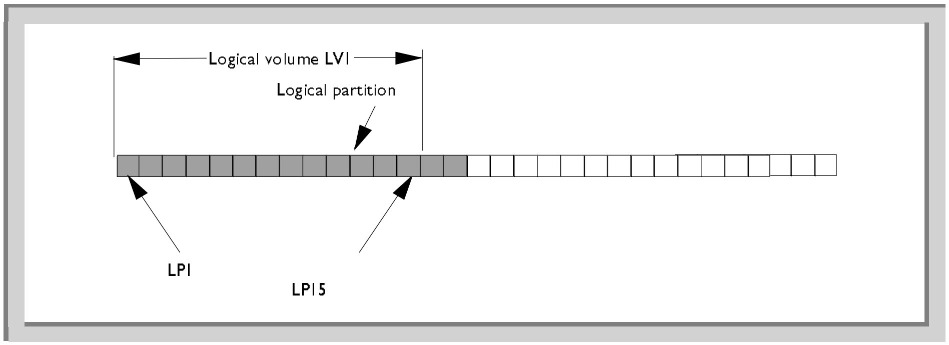


Figure 91. Logical volume LV1

We create a logical volume, LV1, which consists of fifteen logical partitions (see Figure 91). Logical partition LP1 is the first logical partition in the logical volume. Every logical partition points to one physical partition. We create another logical volume, LV2. LV2 has ten logical partitions (see Figure 92).

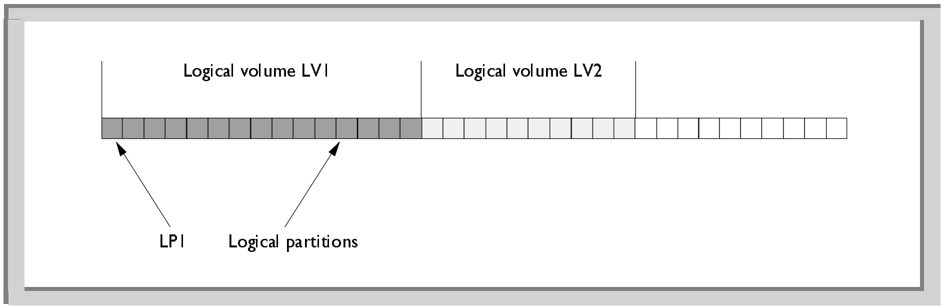


Figure 92. Logical volume LV1+2

We extend logical volume LV1 with ten physical partitions. As you can see, logical volume LV1 is fragmented (see Figure 94).

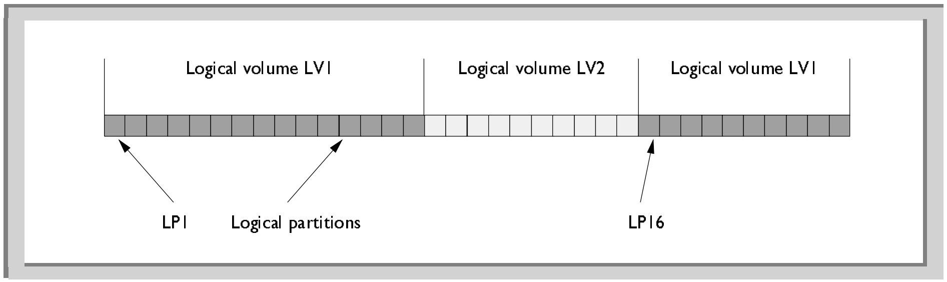


Figure 93. logical volume LV1 and LV2 with LV1 fragmented

Now, we remove LV2 and extend LV1 again (see Figure 94). LV1 seems to be contiguous when you look with `lspv -l hdiskX`, but is not. If you consider that LV2 was a big logical volume, applications doing large sequential I/O on LV1 will suffer from this configuration since there is a huge gap to cross between partition 10 and partion 25 or 25 and 26 (shown in Figure 94). Such sequential I/O cannot be done without many disk head movements.

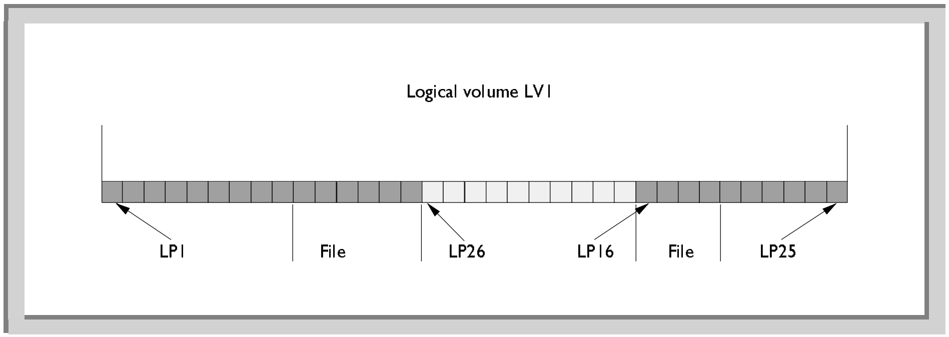


Figure 94. logical volume with LV1 and LV2

7.4.2 Fragmentation example 2

We show another example of a non-recommended way of creating logical volumes. We create a logical volume on a 9.1 GB physical volume in the center location. The logical volume is exactly 108 physical partitions in size (the maximum number of physical partitions available on this location, see Figure 95). It occupies the complete center location of the disk.

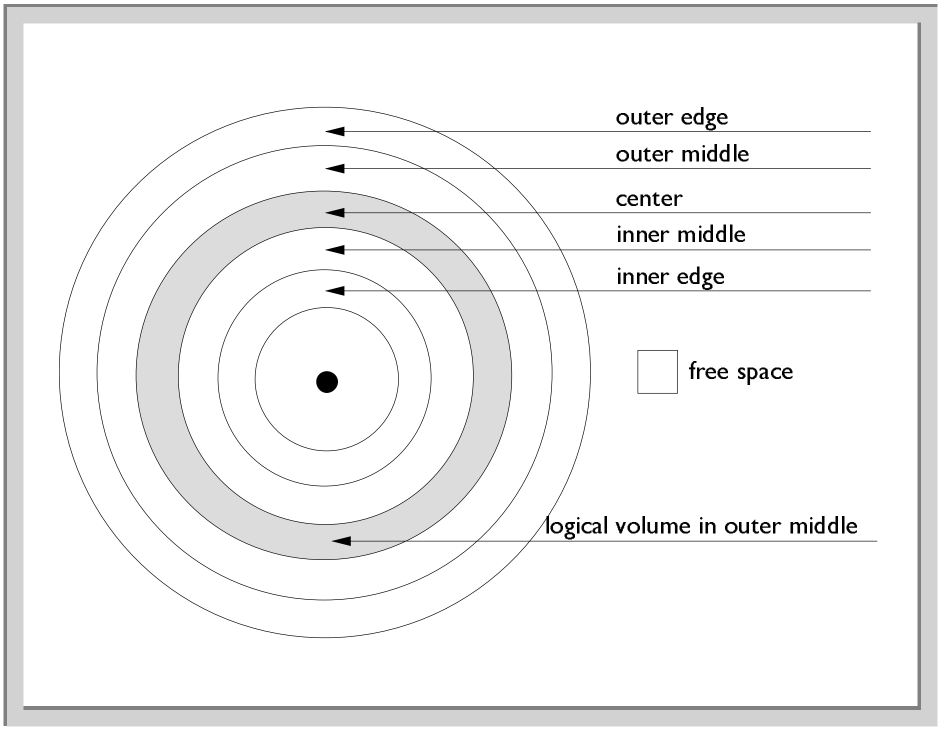


Figure 95. A disk with one logical volume in the center location

Now, we decide that this logical volume is not big enough; so, we extend it with an extra 108 physical partitions. When you look at the distribution with `lspv -l hdiskX`, as shown below, it seems like a contiguous logical volume.

```
brenzef[root] # lspv -l hdisk3
hdisk3:
LV NAME              LPs   PPs   DISTRIBUTION          MOUNT POINT
ronaldlv             216   216   00..108..108..00..00  N/A
```

But, when we look with `lsvg -M ronaldvg`, it is not contiguous at all. The first part looks fine, but after the first 108 physical partitions, you see that it’s not contiguous anymore.

First, the logical partitions are in sequence with the physical partitions, but at physical partition 218, things change. Physical partition 218 is the start of the logical volume ronaldlv. Creating contiguous logical volumes can be forced using map files, which is described in Section 2.2.1.3, “Map files” on page 110.

```
brenzef[root] # lsvg -M ronaldlv
First part
ronaldvg
hdisk3:1-109
hdisk3:110      ronaldlv:109
hdisk3:111      ronaldlv:110
hdisk3:112      ronaldlv:111
hdisk3:113      ronaldlv:112
hdisk3:114      ronaldlv:113
hdisk3:115      ronaldlv:114
hdisk3:116      ronaldlv:115

At the 107th physical partition

hdisk3:216      ronaldlv:215
hdisk3:217      ronaldlv:216
hdisk3:218      ronaldlv:1
hdisk3:219      ronaldlv:2
hdisk3:220      ronaldlv:3
hdisk3:221      ronaldlv:4
hdisk3:222      ronaldlv:5
hdisk3:223      ronaldlv:6
hdisk3:224      ronaldlv:7
hdisk3:225      ronaldlv:8
hdisk3:226      ronaldlv:9
```

Note

Both situations shown in the previous examples will not cause many problems in a random read of a write environment. But, they will cause problems in an environment where big files are read and written sequentially.

7.4.3 The range of the logical volume

If you have more than one physical disk in your volume group, you can spread the logical volume across the physical disks. The `-e` (Range) flag tells the `mklv` command the number of physical volumes to use to spread the logical volume using the regions that provide the best allocation. Configuring a logical volume across more physical disks means that more disk heads can do the work; so, performance will be better. If possible, span a logical volume across more physical disks, keeping in mind that other logical volumes (configured on the same physical disk) may be intensively used. Therefore, make sure that two intensively used logical volumes are not placed on the same physical disk drive. The maximum number of disks you can use for the logical volume is limited by the `-u upperbound` flag. Figure 96 shows you a logical volume that is nicely spread across three disks.

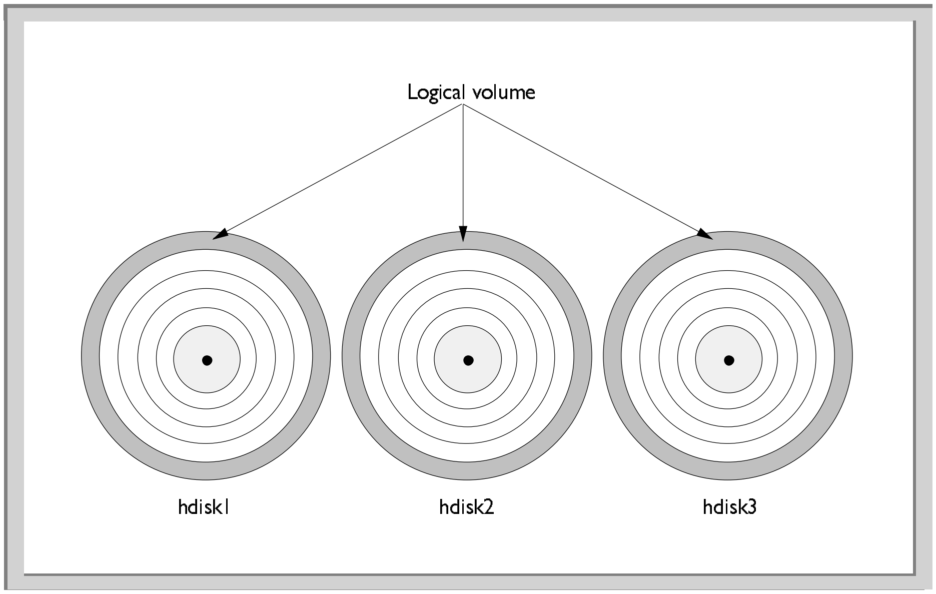


Figure 96. A logical volume across three disks

An example on how to create a logical volume on the outer edge on three disks using an upperbound of three would be:

```
brenzef[root] # mklv -y ronaldlv -a e -e x -u 3 rootvg 30 hdisk0 hdisk1
hdisk2
```

7.4.4 Exact mapping

If you want to specify exactly which physical partitions you want to use for allocation, you must use a map file. With a map file, you tell `mklv` with the `-m <mapfile>` flag which physical volume should be used and which physical partitions should be allocated. The use of map files is described in detail in Section 2.2.1.3, “Map files” on page 110. An example of creating a logical volume using a map file would be:

```
brenzef[root] # mklv -y ronaldlv -m /tmp/ronaldlv.map rootvg 3
```

7.4.5 Write verify

In mirrored configurations, a feature is available for users who want to make sure that data is written and is readable. This option is called write verify. Write verify means that after every write, a read operation is executed to read the data that has just been written. This guarantees the user that the write action took place correctly. You must keep in mind that enabling write verify slows down the I/O process. The write call returns after the written data is read. It returns an error when the read operation fails. Detailed information on how to activate write verify can be found in Section 2.2.3.5, “Write verify” on page 127. An example on how to create a logical volume on the center location with write verify enabled would be:

```
brenzef[root] # mklv -y ronaldlv -a c -v y rootvg 3
```

If you want to activate write verify using the SMIT interface, you should enter:

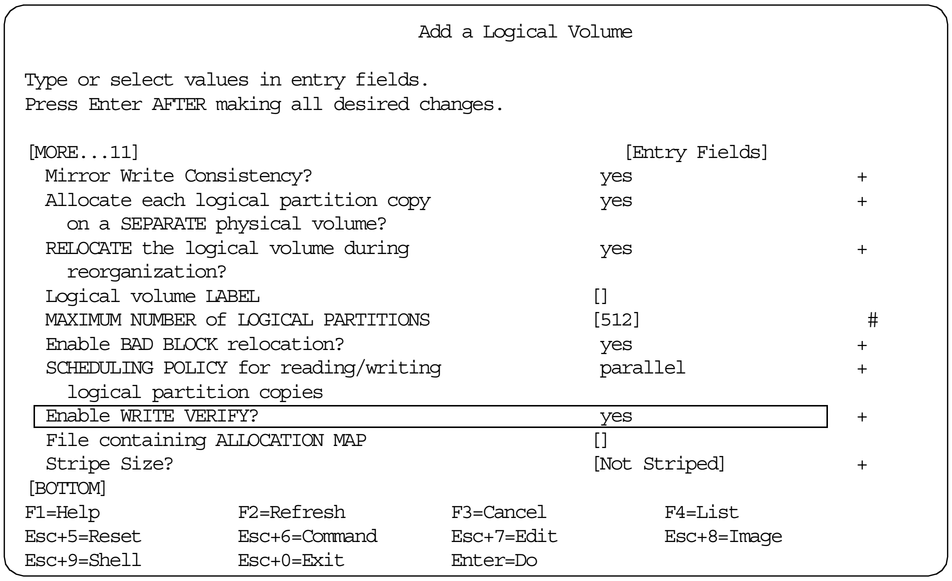
```
brenzef[root] # smitty lvm
```

Select **logical volumes**

Select **Add A Logical Volume**

Enter the name of the volume group

Change `Enable WRITE VERIFY` to `Yes`.



```
                                 Add a Logical Volume

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[MORE...11]                                               [Entry Fields]
  Mirror Write Consistency?                             yes                +
  Allocate each logical partition copy                  yes                +
    on a SEPARATE physical volume?
  RELOCATE the logical volume during                    yes                +
    reorganization?
  Logical volume LABEL                                 []
  MAXIMUM NUMBER of LOGICAL PARTITIONS                 [512]                #
  Enable BAD BLOCK relocation?                          yes                +
  SCHEDULING POLICY for reading/writing                 parallel           +
    logical partition copies
  Enable WRITE VERIFY?                                  yes                +
  File containing ALLOCATION MAP                       []
  Stripe Size?                                         [Not Striped]       +
[BOTTOM]
F1=Help             F2=Refresh          F3=Cancel           F4=List
Esc+5=Reset         Esc+6=Command       Esc+7=Edit          Esc+8=Image
Esc+9=Shell         Esc+0=Exit          Enter=Do
```

Note

Write verify is disabled by default, which is shown as `no` in the smit panels.

7.5 Mirroring

Mirroring is a powerful feature of the AIX LVM. If you create more than one copy, you increase data availability. If one copy fails, the system can continue working using the other copy. But, from a performance point of view, mirroring does not give you the best performance. It is simple to understand that when you have extra copies, these copies must be maintained also, and writing to two or three copies takes longer than writing to one. You might think that reading is much faster on a parallel mirrored configuration than on a one copy configuration because more copies are available for reading. Section 7.5.1, “The mirror read algorithm” on page 321 explains why this is not always the case.

7.5.1 The mirror read algorithm

You might expect that the use of mirroring speeds up the read performance. The following example shows you why this is not always the case

If you have the fix, APAR IX58267/APAR IX59519 on AIX Version 4.1 (later release of AIX does not need any fix), the algorithm reads as follows:

```
number_of_outstanding_ios = I/O count of the disks holding
                            the first mirror in mirror array
mirror_to_read = first copy
for (loop = second copy; loop <= third copy; loop++) {
    if (this loop copy’s outstanding I/Os < number_of_outstanding_ios) {
        number_of_outstanding_ios = this loop copy’s outstanding I/O’s
        mirror_to_read = this mirror (loop)
    }
}
```

Let us assume that an application executes a number of reads. Initially, the first mirror copy will be chosen. But, if the primary disk is busy processing the read the other reads will be scheduled across drives that have the least outstanding requests. When the primary disk has finished, this disk is used again as the first disk to read from.

So, this example shows that reading from a mirrored logical volume is done often using the first copy. If you use sequential mirroring, the primary copy is always the first one used. So, to be able to benefit from mirroring read, you should select the parallel scheduling policy. If you bypass the logical volume manager by using the disk driver directly, you do not benefit from the mirror read performance.

In this new example, we create a logical volume with a hundred 16 MB physical partitions on hdisk1. This logical volume is mirrored to another physical disk hdisk2. We start a `dd` and read from the logical volume and dump the data in /dev/null:

```
dd if=/dev/ronaldlv of=/dev/null bs=102400
```

The `iostat` output shows that the reads are spread across the two physical volumes.

```
brenzef[root] # iostat -d hdisk1 hdisk2
Disks:        % tm_act     Kbps      tps  Kb_read   Kb_wrtn
hdisk1          76.5     4130.0     1032.5       8260         0
hdisk2          71.5     4128.0     1032.0       8256         0
hdisk1          69.3     4095.8     1023.9       8212         0
hdisk2          69.8     4095.8     1024.4       8212         0
hdisk1          75.0     4648.0     1161.5       9296         0
hdisk2          78.5     4646.0     1161.5       9292         0
hdisk1          82.0     5272.0     1318.5      10544         0
hdisk2          85.0     5274.0     1318.0      10548         0
```

If we remove the mirror copy from hdisk2, the data can only be read from hdisk1. The amount of data read in the same time interval is lower that the mirrored situation. The example below show this. In the first example, an average total of 8 Kbps was read, in the second example, 7 Kbps was read in the same time interval. We could read in the same time interval more data from the mirrored configuration than from the non-mirrored configuration.

```
brenzef[root] # iostat -d hdisk1 hdisk2
Disks:        % tm_act     Kbps      tps  Kb_read   Kb_wrtn
hdisk1          99.5     7054.0     1763.5      14108         .0
hdisk2           0.0       0.0       0.0          0         0
hdisk1         100.0     7022.0     1755.5      14044         0
hdisk2           0.0       0.0       0.0          0         0
hdisk1         100.0     7106.0     1776.5      14212         0
hdisk2           0.0       0.0       0.0          0         0
```

7.5.2 Mirror write consistency

The LVM always makes sure that the data is consistent among mirrored copies of a logical volume during normal I/O. If a write is generated, this write will be send to all the mirror copies. But what if the system crashes in the middle of writing such a request to a mirror copy? This is where the mirror write consistency comes in place.

An example of this is when you create your logical volume on the inner edge, and perform many random writes, they will be probably in different logical track groups. Every write will cause the mirror write consistency cache to be updated. This area is located on the outer edge of the disk, which will cause many disk head movements. Having mirror write consistency enabled in this situation will cause a serious performance degradation. So, make sure that in the case of random writes, these areas are close to each other, that is, on the outer edge. Using big physical volumes also has a performance implication. There are more logical track groups per inch since a logical track group has a fixed size of 128 KB. Random writes causes more logical track groups to be updated. Updating more logical track groups causes more writes to the mirror write consistency cache. Again, the outer edge is the most suitable location for these logical volumes.

For a detailed explanation about the mirror write consistency cache, refer to “Mirror Write Consistency” on page 117.

7.6 Reorganizing volume groups

After a logical volume is created, a journaled file system or a RDBMS is put on top. But, often there is a need for more free space because the file system needs to be enlarged, or the RDBMS needs more space to store data. The `chlv` and the `chfs` commands are used to create extra space. The problem, though, is that sometimes there are not enough physical partitions close to the created logical volume available due to creation of other logical volumes on the same disk. The logical volume that needs to be enlarged can not be kept contiguous. This is where the LVM fragmentation comes from. In this section, we describe how to handle LVM fragmentation issues using the `reorgvg` command.

7.6.1 reorgvg

If you are not satisfied with the placement of the logical volumes on the disk, or the logical volumes are fragmented as shown in Figure 89 on page 311 you can reorganize the volume group by running the `reorgvg` command.

The `reorgvg` command reorganizes the placement of allocated physical partitions within a volume group according to the allocation characteristics of each logical volume. You can use the logical volume parameter to reorganize specific logical volumes; highest priority is given to the first logical volume name in the logical volume parameter list, and lowest priority is given to the last logical volume in the parameter list. The volume group must be varied on and must have free partitions before you can use the `reorgvg` command.

The relocatable flag of each logical volume must be set to y with the chlv -r command for the reorganization to take effect for that volume; otherwise, it is ignored. The following example would reorganize logical volume ronaldlv using reorgvg:

```
brenzef[root] # reorgvg rootvg ronaldlv
```

Before the reorgvg command, the placement of the logical volume could look as shown in Figure 97.

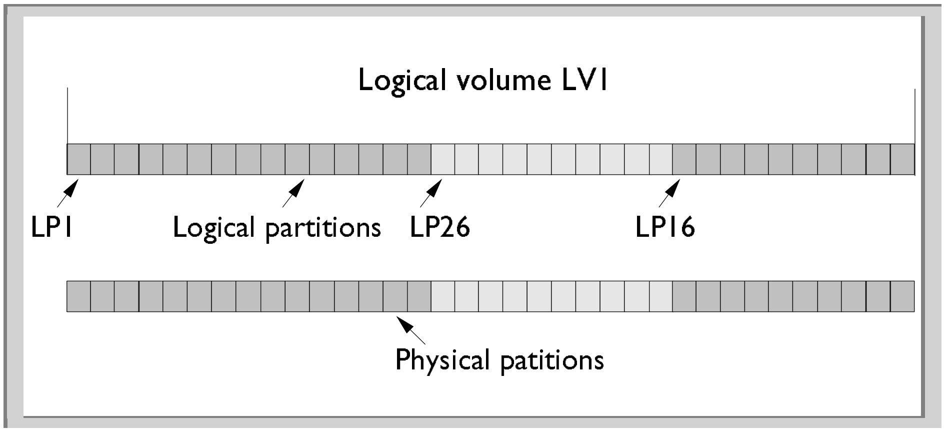


Figure 97. Fragmented logical volume before reorgvg

During reorganization, the logical partitions will be relocated if possible. Figure 98 shows the picture during reorganization.

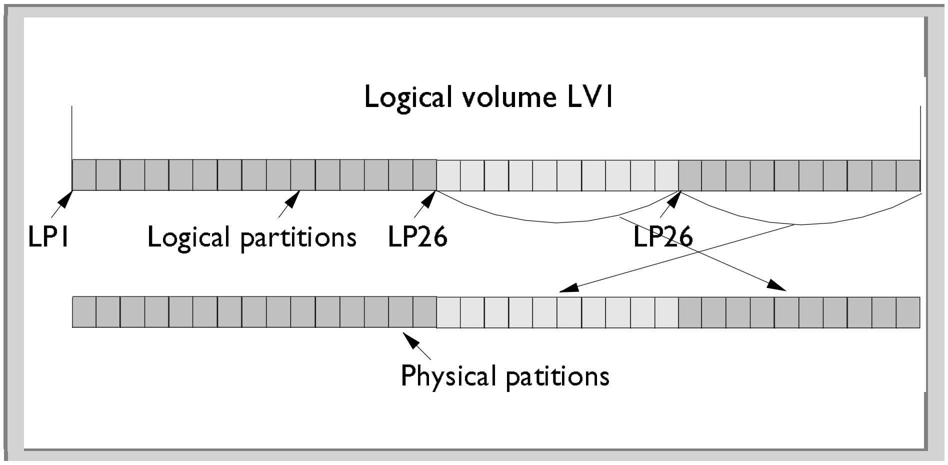


Figure 98. Fragmented logical volume during reorgvg

After reorganization, the logical partitions will be contiguous (if possible).

Things to keep in mind:

- The reorgvg command does not reorganize the placement of allocated physical partitions for any striped logical volumes.
- At least one free physical partition must exist on the specified volume group for the reorgvg command to run successfully (the more, the faster).
- To use this command, you must either have root user authority or be a member of the system group.

7.7 Striping

Striping is a technique for spreading the data in a logical volume across several disk drives in such a way that the I/O capacity of the disk drives can be used in parallel to access data on the logical volume. The primary objective of striping is high-performance reading and writing of large sequential files. Detailed information about how to create and use striped logical volumes can be found in Chapter 3, “Striping” on page 137.

7.7.1 Using disks not equal in size

Creating a striped logical volume on disks that differ in size may cause a performance degradation if you make configuration mistakes. Imagine the situation as shown in Figure 99.

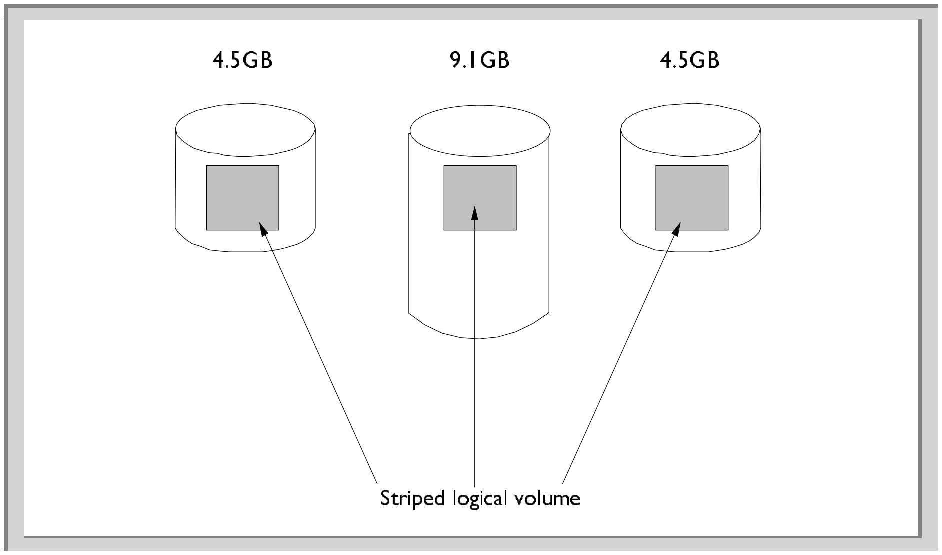


Figure 99. Striping on different size disks

In Figure 99, the striped logical volume consists of three disks; one disk is a 9.1 GB drive; the two other drives are 4.5 GB. The 4.5 GB drives do not have free physical partitions left. Now, we add another logical volume (see Figure 100) on the 9.1GB disk. If this new logical volume is also extensively used, the striped logical volume will suffer. The performance gain you expected from striping the logical volume is now reduced because of the extra I/O that is done on the second logical volume. So, if there is a need for creating an extra logical volume, make sure is does not interfere with the striped logical volume. Another issue is that you cannot extend the striped logical volume using only the 9.1 GB disk because the other two 4.5 GB physical disks do not have free physical partitions available.

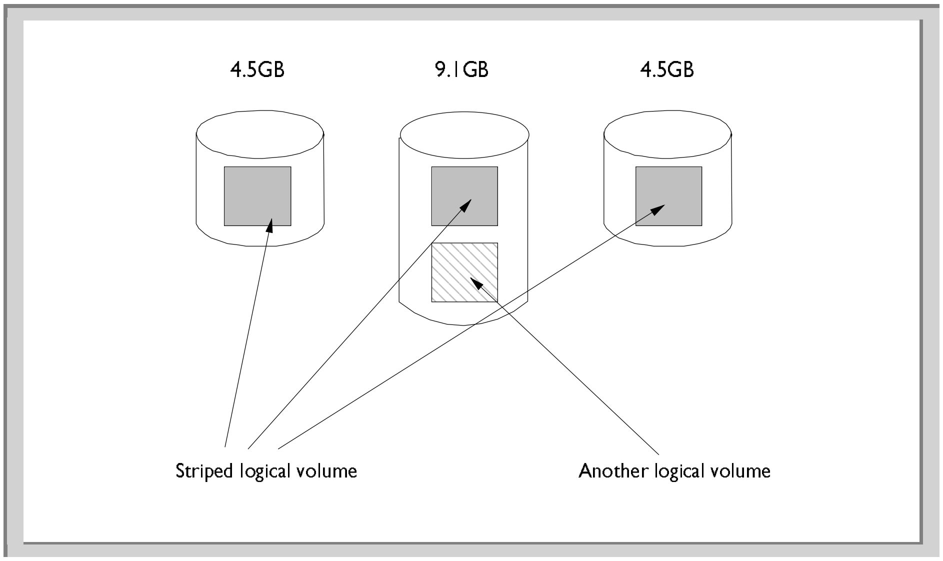


Figure 100. Striping on different size disks with extra logical volume

7.7.2 Striping in a random read/write environment

Initially, striping was developed to get a performance gain in sequential read/write environments. When the right stripesize is chosen, there is a benefit when you use a striped logical volume. In a random read/write environment, the benefit is reduced because a seek time must be added to the time for the I/O operation to complete, just as random read/writes on a “regular disk” would not give you optimal performance.

Note

Striping is not a the ultimate solution for every environment.

7.7.3 The stripe width

The number of disks used for a striped logical volume is also referred to as the *stripe width*. Using a bigger stripe width could increase the performance. But there are some considerations. If you for example striped across 20 disks, you might expect a fast disk I/O response time. However, the adapter and bus must be able to transport the requested reads and writes. The throughput of a disk can be with the current technology as much as 14 to 20 MB. An sequential read from a striped logical volume with a stripe width of 20 could be 20*(14-20 MB); so, a total of 280 MB to 400 MB per second. The PCI busses used these days are not fast enough to transport that amount of data.

Also, the throughput is dependent of the I/O size in an SSA configuration. Since a SSA adapter can only handle a certain amount of I/O’s per second (see Section 7.2.1, “Adapters” on page 305).

7.7.4 pbufs

The Logical Volume Manager (LVM) uses a construct called a "pbuf" to control a pending disk I/O. A single pbuf is used for each I/O request, regardless of the number of pages involved. AIX creates extra pbufs when a new physical volume is added to the system. But, when you use striping, you need more pbufs because one I/O operation causes I/O operations to more disks and, thus, more pbufs. And, when you stripe and mirror, you need even more pbufs. Running out of pbufs reduces performance drastically. When no more pbufs are available, the I/O process is suspended until pbufs are available again. Increasing the number of pbufs can be done with `vmtune`; however, pbufs are pinned so that allocating many pbufs will increase the use of memory. When you suspect you have run out of pbufs, you can do the following using the `crash` command:

On the command line enter:

```
brenzef[root] # crash
```

To find the address where the number of pbuf requests denied are stored enter:

```
> knlist hd_pendqblked
```

```
hd_pendqb: 0x01311738
```

Next do an `od` on that address.

```
> od 01311738
```

```
01311738: 00000027
```

The output shows the number of requests denied since boot time. This example shows that this system had 39 (27 is expressed in hexadecimal) times a request for pbuf was denied because no pbufs where available.

Note

Use the `crash` command with caution in a production environment. Certain `crash` commands could change the contents of the system’s internal memory and could crash the system.

7.7.5 Checking the number of pbufs

The vmtune command can tune the LVM parameter: *hd_pbuf_cnt*. *hd_pbuf_cnt* specifies number of pbufs and pbufs control pending disk I/O requests at the LVM layer. Systems with many sequential I/O requests could benefit from a higher value of *hd_pbuf_cnt*. pbufs are pinned; so, do not set this value too high since the value cannot be lowered without a system reboot. The default value is 96.

Changing the number of pbufs:

```
brenzef[root] # /usr/samples/kernel/vmtune -B 128
```

will set the number of pbufs to 128.

When you issue the command:

```
brenzef[root] # /usr/samples/kernel/vmtune -B 100
```

the system will respond with: hd_pbuf_cnt must be greater than 128.

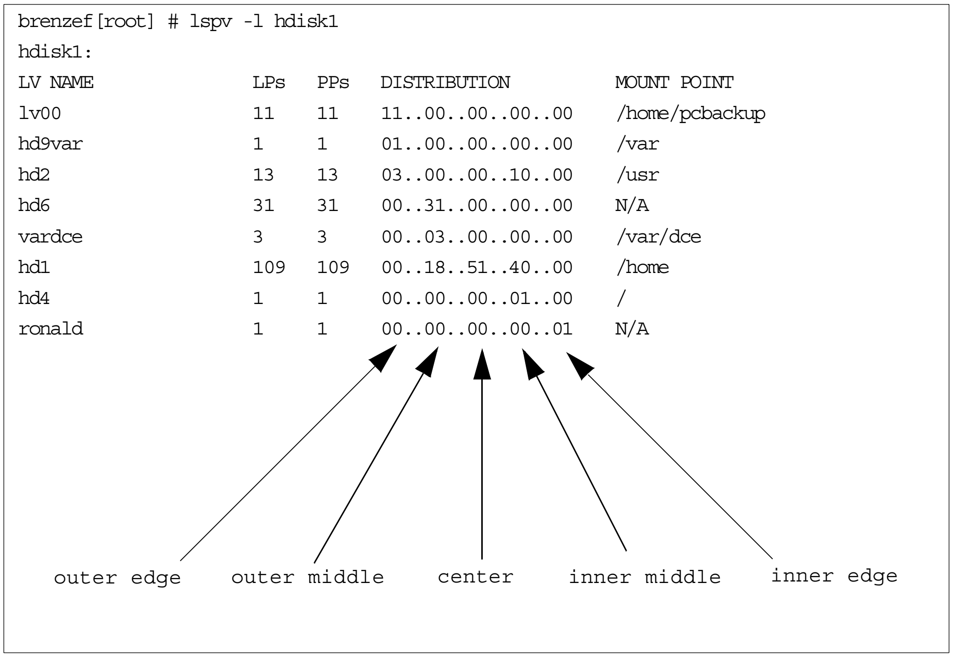
Note

The vmtune command should be used with care. Setting incorrect values could cause unexpected system behavior.

7.8 Finding fragmented logical volumes

Most performance problems arise when the system is used for a while. Applications are added, users are added, and the disk configuration is changed several times. If there is a disk I/O bottleneck, you could look at the disk layout to see if one or more logical volumes are fragmented. If you want to know how the logical volumes are configured across the physical volumes, you can use the following commands.

Assuming you want to investigate hdisk1, run the lspv command to see the disk distribution.



```
brenzef[root] # lspv -l hdisk1
hdisk1:
LV NAME              LPs   PPs   DISTRIBUTION          MOUNT POINT
lv00                 11    11    11..00..00..00..00    /home/pcbackup
hd9var               1     1     01..00..00..00..00    /var
hd2                  13    13    03..00..00..10..00    /usr
hd6                  31    31    00..31..00..00..00    N/A
vardce               3     3     00..03..00..00..00    /var/dce
hd1                  109   109   00..18..51..40..00    /home
hd4                  1     1     00..00..00..01..00    /
ronald               1     1     00..00..00..00..01    N/A
```

Figure 101. Example of a disk distribution

When you look at the DISTRIBUTION column in Figure 101, you see five columns and the location they represent.

The columns show you how many physical partitions are allocated on these locations. There are no fragmented logical volumes on this disk. Another example shows you a physical volume with a fragmented logical volume on it, as you can see in the following example.

```
brenzef[root] # lspv -l hdisk44
hdisk44:
LV NAME              LPs   PPs   DISTRIBUTION          MOUNT POINT
appldata             50    50    50..00..00..00..00    /ronald/appldata
ronaldlv             148   148   50..00..00..46..52    /ronald/database
paging00             14    14    00..14..00..00..00    N/A
```

Looking at the logical volume name `ronaldlv`, you see that 50 physical partitions are located on the outer edge and 46 and 52 on the inner-middle and the inner edge.

Another, more detailed, view of the physical partition's locations can be seen with the `lspv -p` command in the following:

```
brenzef:[root] # lspv -p hdisk0
hdisk0:
PP RANGE  STATE   REGION        LV NAME             TYPE       MOUNT POINT
  1-1     used    outer edge    hd5                 boot       N/A
  2-108   free    outer edge
109-215   used    outer middle  hd6                 paging     N/A
216-216   used    center        hd8                 jfslog     N/A
217-217   used    center        hd4                 jfs        /
218-222   used    center        hd2                 jfs        /usr
223-223   used    center        hd9var              jfs        /var
224-225   used    center        hd3                 jfs        /tmp
226-226   used    center        hd1                 jfs        /home
227-240   used    center        hd2                 jfs        /usr
241-261   used    center        hd6                 paging     N/A
262-294   used    center        hd2                 jfs        /usr
295-297   used    center        raid10              jfs        /raid10
298-299   used    center        hd2                 jfs        /usr
300-322   free    center
323-429   free    inner middle
430-440   used    inner edge    ronaldlv            jfs        N/A
441-537   free    inner edge
```

You can see a logical volume of ten physical partitions with the name `ronaldlv` are allocated at the beginning of the inner edge. The `lslv -p hdisk0 ronaldlv` command can be used to prove this. The `lslv -p hdisk0 ronaldlv` command shows all the locations as shown in the following example. The output consists of five sections. Each section represents a disk region, such as outer edge, inner-middle, and so on. The first block is the outer edge; the last block is the inner edge. The output shows the inner-middle and the inner edge location only.

If the field shows a number, it means the partition is used by the logical volume passed to the `lslv` command. If the field shows FREE, it means that this physical partition is unused. USED means that this physical partition is in use by another logical volume other than the one passed to the `lslv` command. The last column tells you in which physical partition range it is on the physical disk.

```
brenzef:[root] # lslv -p hdisk0 ronaldlv
hdisk0:ronaldolv:N/A
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    323-332
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    333-342
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    343-352
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    353-362
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    363-372
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    373-382
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    383-392
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    393-402
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    403-412
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    413-422
FREE  FREE  FREE  FREE  FREE  FREE  FREE                      423-429

0001  0002  0003  0004  0005  0006  0007  0008  0009  0010    430-439
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    440-449
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    450-459
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    460-469
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    470-479
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    480-489
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    490-499
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    500-509
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    510-519
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE    520-529
FREE  FREE  FREE  FREE  FREE  FREE  FREE  FREE                530-537
```

7.9 Performance considerations

In this section, we give some performance considerations in general in striped situations and in mirroring situations.

7.9.1 General performance considerations

The following are general performance considerations:

1. Spread hot logical volumes across multiple physical volumes so that parallel access is possible.
2. Create your logical volumes big enough to avoid fragmentation when the logical volumes have to be increase later.
3. Place sequential write intensive logical volumes on the outer edge of the physical volumes.
4. Place sequential read-intensive logical volumes in the center of the physical volumes. Place moderate sequential read-intensive logical volumes in the outer-middle of physical volumes and place least sequential read-intensive logical volumes on inner-middle or inner edge of the physical volumes.

5. Make logical volumes contiguous to reduce access time.
6. Set inter-policy to maximum. This will spread each logical volume across as many physical volumes as possible, allowing reads and writes to be shared among several physical volumes.
7. Set write verify to no so that there is no follow-up read performed following a write.
8. Implement striping for heavily used sequential accessed logical volumes.

7.9.2 Performance considerations for striping

The following are performance considerations for striping:

1. The stripe size you use depends on the read/write size of the application. A stripe size of 64 K is a good start.
2. Set max_coalesce equal or as a multiple of the stripe unit size but not smaller.
3. Set minpgahead on 2.
4. Set maxpgahead to 16 * N (where N is the number of disk drives). This causes the page-ahead read to be done in the stripe unit size (64 KB) times the number of disk drives, resulting in the reading of one stripe unit from each disk drive for each read-ahead operation.
5. I/O requests for striped disks (64 KB times the number of disk drives). This is equal to the maxpgahead value.
6. Modify maxfree to accommodate the change in maxpgahead so that the difference between *minfree* and *maxfree* is maxpageahead.
7. 64-byte aligned I/O buffers. If the logical volume will occupy physical drives that are connected to two or more disk adapters, the I/O buffers used should be allocated on 64-byte boundaries. This avoids having the LVM serialize the I/Os to the different disks.
8. If the striped logical volumes are raw logical volumes, and the writes larger than 1.125 MB are being done to these striped raw logical volumes, increasing the *lvm_bufcnt* parameter of `vmtune` might increase throughput of the write activity.
9. Do not create intensively used logical volumes on a disk with a striped logical volume on it.
10. Check with crash if you have enough pbufs for a striped or mirrored/striped logical volume (see Section 7.7.5, “Checking the number of pbufs” on page 329).
11. Tune the number of pbufs with `vmtune -B`.

7.9.3 Performance considerations for mirroring

The following are performance considerations for mirroring:

1. Do not use mirroring, if possible, since writing a mirror performs multiple writes that will affect performance in writing. If mirroring is needed, set the scheduling policy to parallel and allocation policy to strict. The parallel scheduling policy will enable reading from the disk that has the least outstanding requests, and strict allocation policy allocates each copy on separate physical volume(s).
2. Locate intensive mirrored logical volumes on the outer edge because, in that situation, the mirror write consistency cache must be updated on a regular basis.

7.10 The application layer

We will now concentrate on the last layer between the use and the disk, the application layer. This layer can be a journaled file system or an RDBMS using the RAW logical volume beneath it. Since there is a complete redbook covering performance in RDBMS, *Database Performance on AIX in DB2 UDB and Oracle Environments*, SG24-5511, we will focus on the JFS aspect of performance.

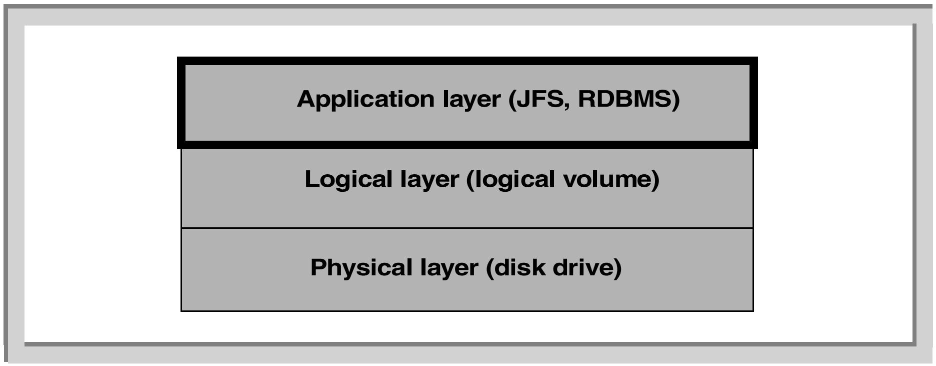

Application layer (JFS, RDBMS)
Logical layer (logical volume)
Physical layer (disk drive)

Figure 102. LVM components, the application layer

7.10.1 The journaled file system

Journaled file systems have several options that affect performance, and most of them are covered in the *AIX Performance and Tuning Guide*, SC23-2365. We will discuss here only the JFS fragmentation and the use of more JFS log logical volumes.

7.10.1.1 Journaled file system fragmentation

While an AIX file is conceptually a sequential and contiguous string of bytes, the physical reality may be very different. Fragmentation may arise from multiple extensions to logical volumes as well as an allocation / release / reallocation activity within a file system. We say a file system is fragmented when its available space consists of large numbers of small chunks of space, thus, making it impossible to write out a new file in contiguous blocks.

Access to files in a highly-fragmented file system may result in a large number of seeks and longer I/O response time (seek latency dominates I/O response time). For example, if the file is accessed sequentially, a file placement that consists of many, widely separated chunks requires more seeks than a placement that consists of one large, contiguous chunk. If the file is accessed randomly, a placement that is widely dispersed requires longer seeks than a placement in which the file blocks are close together.

The effect of a file placement on I/O performance diminishes when the file is buffered in memory. When a file is opened in AIX, it is mapped to a persistent data segment in virtual memory. The segment represents a virtual buffer for the file. The file blocks map directly to segment pages. The VMM manages the segment pages, reading file blocks into segment pages upon demand (as they are accessed). There are several circumstances that cause the VMM to write a page back to its corresponding block in the file on disk; but, in general, the VMM keeps a page in memory if it has been accessed recently. Thus, frequently accessed pages tend to stay in memory longer, and logical file accesses to the corresponding blocks can be satisfied without physical disk accesses.

At some point, the user or system administrator may choose to reorganize the placement of files within logical volumes and the placement of logical volumes within physical volumes to reduce fragmentation and to more evenly distribute the total I/O load. More about `defragfs` can be found in Section 7.10.1.3, “defragfs” on page 338.

7.10.1.2 Finding fragmented files.

When you think your file or files are fragmented, you can use the `fileplace` command to investigate the file. The `fileplace` command displays the placement of a specified file within the AIX logical or physical volumes containing the file.

By default, the `fileplace` command lists to standard output the ranges of logical volume fragments allocated to the specified file.

The order in which the logical volume fragments are listed corresponds directly to their order in the file. A short header indicates the file size (in bytes), the name of the logical volume in which the file lies, the block size (in bytes) for that volume, the fragment size in bytes, and the compression mechanism, indicating if the file system is compressed or not.

Occasionally, portions of a file may not be mapped to any fragments in the volume. These areas, whose size is an integral number of fragments, are implicitly zero-filled by the file system. The `fileplace` command will indicate which areas in a file have no allocated fragments.

Optionally, the `fileplace` command will also display:

- Statistics indicating the degree to which the file is spread within the volume.
- The indirect block addresses for the file.
- The file placement on physical (as opposed to logical) volume, for each of the physical copies of the file.

The syntax of `fileplace` is:

```
fileplace [-l] [-p] [-i] [-v] <filename>

        -l:  display logical  blocks

        -p:  display physical blocks

        -i:  display indirect blocks

        -v:  verbose mode (show efficiency and sequentiality)
```

An example of the `fileplace` command would be:

```
brenzef[root] # fileplace -v wtmp

File: wtmp  Size: 56000 bytes  Vol: /dev/hd9var
Blk Size: 4096  Frag Size: 512  Nfrags: 112   Compress: no
Inode: 27  Mode: -rw-rw-r--  Owner: adm  Group: adm

  Logical Fragment
  ----------------
  0002952                         8 frags     4096 Bytes,   7.1%
  0002975-0002983                16 frags     8192 Bytes,  14.3%
  0003008-0003024                24 frags    12288 Bytes,  21.4%
  0003045-0003053                16 frags     8192 Bytes,  14.3%
  0003085                         8 frags     4096 Bytes,   7.1%
  0003107                         8 frags     4096 Bytes,   7.1%
  0002768                         8 frags     4096 Bytes,   7.1%
  0003115                         8 frags     4096 Bytes,   7.1%
  0003179                         8 frags     4096 Bytes,   7.1%
  0003576                         8 frags     4096 Bytes,   7.1%

  112 frags over space of 816 frags:  space efficiency = 13.7%
  10 fragments out of 112 possible:  sequentiality = 91.9%
```

The output of the `fileplace` command shows the location of the logical fragments on the file system. The wtmp file in /var/adm is spread across 816 fragments but only uses 112 fragments, which is not space efficient. The space efficiency is then 13.7 percent. The fragments, however, are placed sequentially for 91.9 percent.

Another example is as follows:

```
brenzef[root] # fileplace -v /smit.log

File: /smit.log  Size: 73378 bytes  Vol: /dev/hd4
Blk Size: 4096  Frag Size: 4096  Nfrags: 18   Compress: no
Inode: 71  Mode: -rw-r--r--  Owner: root  Group: system

  Logical Fragment
  ----------------
  0000018                         1 frags     4096 Bytes,   5.6%
  0000607                         1 frags     4096 Bytes,   5.6%
  0000621                         1 frags     4096 Bytes,   5.6%
  0000619-0000620                 2 frags     8192 Bytes,  11.1%
  0000624-0000625                 2 frags     8192 Bytes,  11.1%
  0000623                         1 frags     4096 Bytes,   5.6%
  0000628-0000630                 3 frags    12288 Bytes,  16.7%
  0000633-0000634                 2 frags     8192 Bytes,  11.1%
  0000631-0000632                 2 frags     8192 Bytes,  11.1%
  0000636-0000638                 3 frags    12288 Bytes,  16.7%

  18 frags over space of 621 frags:  space efficiency = 2.9%
  10 fragments out of 18 possible:  sequentiality = 47.1%
```

The smit.log is a file that is often written to and could be fragmented. The `fileplace` command shows that this file consists of 18 fragments and is placed across 621 fragments, which gives a space affiancing of 2.4 percent. The start of this file is on logical fragment 18, and the end of the file is logical fragment 618. Because of that, this file is 47.1 percent sequential.

7.10.1.3 defragfs

The `defragfs` command increases a file system's contiguous free space by reorganizing a file's allocations to be contiguous rather than scattered across the disk. You can specify the name of the file system, as stated in /etc/filesystems, or you can specify the name of the logical volume, such as `/data` or `/dev/datalv`.

The `defragfs` command is written for de-fragmenting regular and compressed file systems. You can use the `defragfs` command to increase contiguous free space in non-fragmented file systems.

Note

Some free space is necessary to allow the `defragfs` command to operate.

Running `defragfs` does not solve the problem of fragmented files. The `Defragfs` command only reorganizes the free disk blocks so more contiguous free space is available. An example of running `defragfs -r` would be:

```
brenzef[root] # defragfs -r /ronald
statistics before running defragfs:
number of free fragments 12557
number of allocated fragments 3827
number of free spaces shorter than a block 23
number of free fragments in short free spaces 53

statistics after running defragfs:
number of free spaces shorter than a block 2
number of free fragments in short free spaces 4

other statistics:
number of fragments moved 229
number of logical blocks moved 55
number of allocation attempts 49
number of exact matches 29
```

The `-r` flag does not change anything; so, no reorganizing is done. It only reports the current status and what would be done if `defragfs` was run. When we run `defragfs` without any flag, the result is:

```
brenzef[root] # defragfs /ronald
statistics before running defragfs:
number of free fragments 12557
number of allocated fragments 3827
number of free spaces shorter than a block 24
number of free fragments in short free spaces 53

statistics after running defragfs:
number of free spaces shorter than a block 4
number of free fragments in short free spaces 12

other statistics:
number of fragments moved 261
number of logical blocks moved 56
number of allocation attempts 48
number of exact matches 28
```

As mentioned previously, `defragfs` will not defragment your files on the file system, it will only defragment the free space within the file system. If you have a file that is sequentially read and written, and discovered that it is fragmented, the only way to de-fragment this file is to back up the file and restore it after you have run a `defragfs` to make sure the free space is not fragmented.

7.10.1.4 Creating an additional log logical volume

In some cases, it could be useful to create extra log logical volumes for journaled file systems in a volume group. If you have a JFS that is write extensive, the use of the log logical volume might cause an I/O bottleneck if it is placed on the same disk. So, in such cases, it is useful to create another log logical volume for that file system on another disk. This will spread the amount of I/O. The following are the steps used to change the log logical volume for the file system data-fs. The data-fs file system is located on hdisk2.

1. Unmount the file system for which you want to create the new log logical volume:

```
brenzef[root] # umount /data-fs
```

2. Create a new log logical volume:

```
brenzef[root] # mklv -t jfslog -y datafsloglv datavg 1 hdisk1
```

3. Format the log:

```
brenzef[root] # /usr/sbin/logform /dev/datafsloglv
```

4. Modify /etc/filesystems with `chfs`:

```
brenzef[root] # chfs -a log=/dev/datafsloglv /data-fs
```

In /etc/filesystems, the following stanza should be visible:

```
/data-fs:
dev= /dev/datalv
vfs= jfs
log= /dev/datafsloglv
mount= true
options= rw
account= false
```

You can also check if the logical volume control block is updated by entering the following command:

```
brenzef[root] # getlvcb -AT datalv
```

The output is shown as follows:

```
brenzef[root] # getlvcb -AT datalv
        AIX LVCB
        intrapolicy = m
        copies = 1
        interpolicy = m
        lvid = 000416316c642fa6.4
        lvname = lv00
        label = /datalv
        machine id = 416314C00
        number lps = 2
        relocatable = y
        strict = y
        stripe width = 0
        stripe size in exponent = 0
        type = jfs
        upperbound = 32
        fs = log=/dev/datafsloglv:options=rw:account=false
        time created  = Fri Sep 24 09:05:57 1999
        time modified = Fri Sep 24 09:06:01 1999
```

4. Mount the changed file system:

```
brenzef[root] # mount /data-fs
```

Placing the log logical volume on a physical volume different from your most active file system logical volume will increase parallel resource usage. You are allowed to have a separate log for each file system. When creating your logical volumes, remember that the performance of drives differs. Try to create a logical volume for a hot file system on a fast drive. and do not put this log logical volume on another hot physical disk.

Note

Do not change the /etc/filesystems file with `vi` or another editor. These changes will not be written into the VGDA and may cause problems after the volume group is exported and imported (for example in HACMP environments).

Mounting a file system using the `-o nointegrity` command could improve performance substantially. If you have a file system where only temporary files are stored, and you do not care about the integrity of these files (because they are just temporary), you could use this flag to improve JFS performance. If the system crashes and was not able to write the buffers to disk, a `fsck` should be run to ensure the file system integrity.

Mounting file system /ronald with no integrity would be:

```
brenzef[root] # mount -o nointegrity /ronald
```

7.10.2 JFS compression

File systems using data compression have the following performance consideration:

In order to perform data compression, approximately 50 CPU cycles per byte are required, and about 10 CPU cycles per byte for decompression. Therefore, compression gives you more CPU load, which can be a problem in situations where there already is a CPU bottleneck. Check how much time the kernel process `jfsc` uses when you suspect that compression causes a performance bottleneck.

Note

The kernel process, `jfsc`, only comes to live when you create a compressed file system.

To find the kernel process, do the following:

```
brenzef[root] # pstat |grep jfsc
```

If `jfsc` is running, the output will be something like the following:

```
68 a    4486      0   2bba     0     0     1  jfsc
```

Convert the process id 4486 to decimal, which would be 17542:

```
brenzef[root] # ps aux |grep 17542
```

The time column shows you the CPU time used.

7.11 Filemon example

The `filemon` command may be the first performance tool to use to investigate I/O related performance. The syntax of the command can be found in the AIX documentation or in the Performance and Tuning Guide. This section provides a practical example with a detailed explanation for the fields included in the output file.

In our example, we will create a dump of the /var file system into a file, /inner/datafile.1. We want to run `filemon` to see where the huge amounts of I/O's are going to. We run the command:

```
brenzef[root] filemon -O all -o filemon.out
```

The `filemon` command tells us what to do to stop `filemon` from monitoring.

Enter the `trcstop` command to complete `filemon` processing.

Just after issuing the `filemon` command, we start a `dd` copy from /dev/hd9var to /inner/datafile.1:

```
brenzef[root] # dd if=/dev/hd9var of=/inner/datafile.1
```

As soon as the copy has finished, we stop `filemon` by entering:

```
brenzef[root] # trcstop
```

`Filemon` responds by issuing the message:

```
[filemon: Reporting started]
[filemon: Reporting completed]
[filemon: 11.232 secs in measured interval]
```

Now, we can examine the output file, filemon.out, to see what information is collected. The exhibits shown in this topic are parts of the filemon.out file. The complete file can be found in Appendix D, “The filemon.out file” on page 369.

```
brenzef[root] # vi filemon.out
```

The header

The header shows the following data:

- The time and date

- The system name
- The measured interval
- The CPU utilization

```
Mon Oct  4 15:19:46 1999
System: AIX brenzef Node: 4 Machine: 000416314C00
11.323 secs in measured interval
Cpu utilization:  24.6%
```

Most active files

The next section (*Most Active Files Report*), shows the most active files during this measurement. It tells you:

- How many MBs are transferred from or to that particular file.
- The number of times the file was opened.
- The number of read system calls made against the file.
- The number of write system calls made against the file.
- The name of the file.
- The name of the volume which contains the file and the files i-node number.

In the example, the following data was gathered:

```
 Most Active Files
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                 volume:inode
------------------------------------------------------------------------
  16.0      1  32769      0  hd9var
  16.0      1      0  32768  datafile.1           /dev/inner:17
   0.1     12     14      0  group                /dev/hd4:4110
   0.0      1     11      0  cmdnfs.cat           /dev/hd2:41871
   0.0      1      6      0  user                 /dev/hd4:39
   0.0      1      6      0  limits               /dev/hd4:30
   0.0      1      6      0  environ              /dev/hd4:26
   0.0      8      7      0  passwd               /dev/hd4:4145
```

In the above screen, we can see that 16 MB is read from file hd9var. This file is opened once, and 32769 read system calls where made against that file. No write system calls where made against the hd9var file. Also, 16 MB is transferred from or to file datafile.1. The file was opened once; no read system calls where made against the file datafile.1, but 32768 write system calls where made against that file. The file, datafile.1, is located on /dev/inner and has i-node number 17. If we want to confirm that i-node 17 is indeed datafile.1, we can use the `ncheck` command. The `ncheck` command scans the i-node table and looks for the i-node supplied and returns the filename:

```
brenzef[root] # ncheck -i 17 /dev/inner
```

The `ncheck` command returns:

```
/dev/inner:
17      /datafile.1
```

Most active segments

The next section in the `filemon` output file is “Most Active Segments”. The most active segment section gives you information about:

- The total amount of MB transferred from and to the segment.
- The number of 4096-byte pages read into the segment from disk.
- The number of 4096-byte pages written from segment to the disk.
- The internal segment ID.
- The type of the segment.
- The name of the volume.

```
Most Active Segments
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs  segid  segtype           volume:inode
------------------------------------------------------------------------
  16.0      0   4092   3f66  page table
   0.0      0      6   b877  log
   0.0      0      2   41e8  page table
   0.0      0      1   86f1  page table
   0.0      0      1   af54  log
```

We can see that, in total, 16 MB have been transferred from the segment with the ID 3f66. 4092 pages of 4096-bytes where written from that segment to the disk.

Most active logical volume

The next section in the filemon.out file is “Most Active Logical Volumes”. This section tell us which logical volumes where used the most during the measured time interval. It show us:

- The utilization of the logical volume (fraction of time busy)
- The number of 512-byte blocks read from the logical volume
- The number of 512-bytes blocks written to the logical volume
- The number of KB per second transferred from and to the logical volume

- The name of the volume
- The description of the volume

```
Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume              description
------------------------------------------------------------------------
  0.52      0  32744 1445.9  /dev/inner          /inner
  0.51  32776     16 1448.0  /dev/hd9var         /var Frag_Sz.= 512
  0.01      0     48    2.1  /dev/hd8            jfslog
  0.00      0      8    0.4  /dev/loglv00        jfslog
```

As shown above, the file system /inner is utilized for 52 percent, which is shown as 0.52 in the exhibit. The /var file system is also used for 51 percent, which is shown as 0.51 in the exhibit. 32776 512-byte block are read from /dev/hd9var, which is about 16 MB.

Most active physical volumes

The next section in the filemon.out file is the “Most Active Physical Volumes” section. This section provides the following information:

- The utilization of the physical volume
- The number of 512-byte blocks read form the physical volume
- The number of 512-byte blocks written to the physical volume
- The amount of data per second read from and written to the physical volume
- The name of the volume
- The description of the volume

```
Most Active Physical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume              description
------------------------------------------------------------------------
  0.40      0  32867 1451.4  /dev/hdisk2         N/A
  0.40      0  32867 1451.4  /dev/hdisk1         N/A
  0.27  32768     64 1449.8  /dev/hdisk0         N/A
  0.00      0      8    0.4  /dev/hdisk3         N/A
```

The exhibit above shows a 40 percent utilization, shown as 0.40 of hdisk1 and hdisk2. Hdisk0 is 27 percent utilized. From this sample, we could conclude that hdisk1 and hdisk2 are mirrored (because the amount of data written to both volumes is equal). The `lslv -l` inner shows that this is true:

```
brenzef[root] # lslv -l inner
inner:/inner
PV                COPIES         IN BAND        DISTRIBUTION
hdisk1            054:000:000    98%            000:000:000:053:001
hdisk2            054:000:000    98%            000:000:000:053:001
```

Detailed file statistics

The next section is “Detailed File Statistics”. It gives you detailed information on file usage. The following information is gathered by `filemon`:

- The filename (if possible)
- The name of the logical volume
- The i-node number for files in a file system
- The number of times the file was opened during measurement
- The total amount of bytes transferred from and to the file
- The number of read calls issued against the file
- The read transfer size statistics (avg/min/max/sdev) in bytes
- The read response statistics (avg/min/max/sdev) in milliseconds
- The number of write calls issued against the file
- The write transfer size statistics (avg/min/max/sdev) in bytes
- The write response statistics (avg/min/max/sdev) in milliseconds
- The write response time statistics
- The number of lseek subroutine calls

The sample of the filemon.out file shows you the statistics for the file accessed during the measurement.

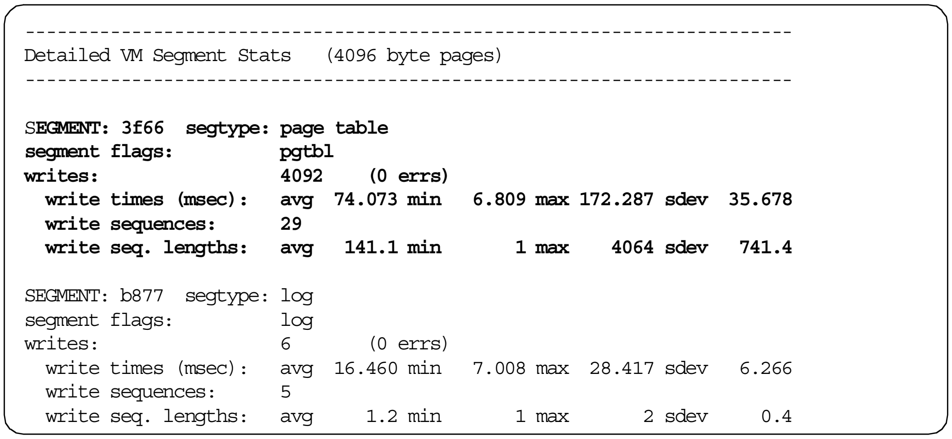
```
------------------------------------------------------------------------
Detailed File Stats
------------------------------------------------------------------------
FILE: /dev/hd9var
opens:                  1
total bytes xfrd:       16777728
reads:                  32769   (0 errs)
  read sizes (bytes):   avg   512.0 min     512 max     512 sdev     0.0
  read times (msec):    avg   0.112 min   0.006 max  30.543 sdev   0.604

FILE: /inner/datafile.1  volume: /dev/inner (/inner)  inode: 17
opens:                  1
total bytes xfrd:       16777216
writes:                 32768   (0 errs)
  write sizes (bytes):  avg   512.0 min     512 max     512 sdev     0.0
  write times (msec):   avg   0.029 min   0.008 max   1.459 sdev   0.048
lseeks:                 1
```

When we look to the file /dev/hd9var, it is opened one time. 16777728 bytes where transferred from and to the file, and 32769 read system calls where made against this file. The read size averaged 512-bytes. The maximum read size was 512-bytes, as was the maximum read size. The average read response was 0.112 milliseconds. The minimum read response was 0.006 milliseconds, and the maximum was 30.543 milliseconds.

Detailed Virtual Memory Segment Statistics

This section covers detailed virtual memory statistics. It gives you detailed information on virtual memory segment usage. The following information is gathered by `filemon`:

- The internal AIX segment ID number
- The type of the segment
- The various segment attributes
- The name of the logical volume (persistent segments only)
- The number of 4096-byte pages read into the segment
- The read response-time statistics (avg/min/max/sdev) in milliseconds
- The number of read sequences (a string of pages that are read in)
- The statistics describing the length of the read sequences
- The number of pages written from the segment (page out)
- The write response time statistics.
- The number of write sequences (a string of pages that are written)
- The statistics that describe the length of the write sequences



```
----------------------------------------------------------------------
Detailed VM Segment Stats   (4096 byte pages)
----------------------------------------------------------------------

SEGMENT: 3f66  segtype: page table
segment flags:          pgtbl
writes:                 4092    (0 errs)
  write times (msec):   avg  74.073 min   6.809 max 172.287 sdev  35.678
  write sequences:      29
  write seq. lengths:   avg   141.1 min       1 max    4064 sdev   741.4

SEGMENT: b877  segtype: log
segment flags:          log
writes:                 6       (0 errs)
  write times (msec):   avg  16.460 min   7.008 max  28.417 sdev   6.266
  write sequences:      5
  write seq. lengths:   avg     1.2 min       1 max       2 sdev     0.4
```

The screen shot above shows another part of the filemon.out file, the virtual memory statistics part. From segment 3f66, 4092 4K pages are written (page out). The write response time has an average of 74.073 milliseconds. The fastest write was 6.809 milliseconds, and the slowest write was 172.287 milliseconds. The standard deviation was 35.678 milliseconds. The number of write sequences was 29, meaning that the 4092 writes where done in 29 sequences. The average write sequence length was 1411 pages; the minimum 1 page, and the maximum 4064 pages. The standard deviation in this measurement was 741.4 pages.

Detailed logical volume statistics

The next section in the filemon.out file shows us detailed information about the logical volume usage. This section shows the following information:

- The name of the volume
- The description of the volume
- The number or reads against the volume
- The read size statistics (avg/min/max/sdev) in units of 512-byte blocks
- The read response time (avg/min/max/sdev) in milliseconds
- The number of read sequences
- The length of the read sequences
- The number of writes against the volume
- The write transfer size
- The write response time
- The number of write sequences

- The length of the write sequences
- The percentage of the reads and writes that required seeks
- The seek distance in units of 512-byte blocks
- The seek distance in units of disk cylinders
- The time to the next read or write request
- The troughput in KB per second
- The utilization of the volume

The following shows the part from the filemon.out file:

```
------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------
VOLUME: /dev/inner  description: /inner
writes:                 1118    (0 errs)
  write sizes (blks):   avg    29.3 min       8 max      32 sdev     6.6
  write times (msec):   avg  74.969 min   6.785 max 172.230 sdev  35.819
  write sequences:      9
  write seq. lengths:   avg  3638.2 min       8 max    8192 sdev  3189.3
seeks:                  9       (0.8%)
  seek dist (blks):     init   9064,
                        avg  1164.0 min       8 max    8216 sdev  2674.3
time to next req(msec): avg   7.877 min   0.007 max 2843.664 sdev  84.943
throughput:             1445.9 KB/sec
utilization:            0.52

VOLUME: /dev/hd9var  description: /var Frag_Sz.= 512
reads:                  4097    (0 errs)
  read sizes (blks):    avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):    avg   0.793 min   0.363 max  30.601 sdev   1.561
  read sequences:       2
  read seq. lengths:    avg 16388.0 min    9736 max   23040 sdev  6652.0
writes:                 2       (0 errs)
  write sizes (blks):   avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):   avg  15.011 min  10.704 max  19.318 sdev   4.307
  write sequences:      2
  write seq. lengths:   avg     8.0 min       8 max       8 sdev     0.0
seeks:                  3       (0.1%)
  seek dist (blks):     init      0,
                        avg  8384.7 min    4785 max   12569 sdev  3204.6
time to next req(msec): avg   2.149 min   0.012 max 2876.785 sdev  44.934
throughput:             1448.0 KB/sec
utilization:            0.51
```

Looking at volume /dev/inner, we see that 1118 writes where made against the volume. The average write size was 29,3 blocks. Minimum write size was 8 blocks, and the maximum 32 blocks. The standard deviation is 6, meaning that the average write size is high. The average write response time is 74.969 milliseconds; the minimum response time is 6.895 milliseconds, and the standard deviation is 35.819 milliseconds, meaning that the response times are average. The number of seeks is 9. The number of write sequences is also 9, meaning that for every write sequence, a seek was issued. The seek distance (seek dist) was 9064. Seek distance is in units of 512-byte blocks. The time to the next request is an average of 7.877 milliseconds, maximum 0.007 milliseconds, and maximum 2843.664 milliseconds. It tells you the rate at which the volume is being accessed; so, on average, every 7.877 milliseconds, a request was issued. The volume throughput in this measurement was 1445.9 KB per second, which gives a utilization of 52 percent, shown as 0.52.

Detailed physical volume statistics

The next section in the filemon.out file shows detailed information about the logical volume usage. This section shows the following information:

- The name of the volume
- The description of the volume
- The number or reads against the volume
- The read size statistics (avg/min/max/sdev) in units of 512-byte blocks
- The read response time (avg/min/max/sdev) in milliseconds
- The number of read sequences
- The length of the read sequences
- The number of writes against the volume
- The write transfer size
- The write response time
- The number of write sequences
- The length of the write sequences
- The percentage of the reads and writes that required seeks
- The seek distance in units of 512-byte blocks
- The seek distance in units of disk cylinders
- The time to the next read or write request
- The troughput in KB per second
- The utilization of the volume

The following shows the last part from the filemon.out file.

```
------------------------------------------------------------------------
Detailed Physical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------
VOLUME: /dev/hdisk2  description: N/A
writes:                 511     (0 errs)
  write sizes (blks):   avg    64.3 min       1 max     128 sdev    53.5
  write times (msec):   avg  16.386 min   0.005 max  59.821 sdev  14.104
  write sequences:      228
  write seq. lengths:   avg   144.2 min       1 max     768 sdev   191.2
seeks:                  228     (44.6%)
  seek dist (blks):     init 5289064,
                        avg 4573452.9 min       1 max 5314558 sdev 1818766.1
  seek dist (%tot blks):init 60.00842,
                        avg 51.88927 min 0.00001 max 60.29767 sdev 20.63527
time to next req(msec): avg  20.418 min   0.009 max 2843.826 sdev 137.186
throughput:             1451.4 KB/sec
utilization:            0.40

VOLUME: /dev/hdisk1  description: N/A
writes:                 515     (0 errs)
  write sizes (blks):   avg    63.8 min       1 max     128 sdev    53.3
  write times (msec):   avg  15.540 min   0.005 max  70.753 sdev  14.057
  write sequences:      229
  write seq. lengths:   avg   143.5 min       1 max     768 sdev   190.3
seeks:                  229     (44.5%)
  seek dist (blks):     init 5289064,
                        avg 4600014.3 min       1 max 5314558 sdev 1790467.7
  seek dist (%tot blks):init 60.00842,
                        avg 52.19063 min 0.00001 max 60.29767 sdev 20.31421
time to next req(msec): avg  20.417 min   0.012 max 2843.725 sdev 137.179
throughput:             1451.4 KB/sec
utilization:            0.4
```

The last section gives you detailed information about physical volume statistics. You can see the volume (/dev/hdisk2), which is the most active during the measurement. 515 reads where made against this volume. The average write size was 64.3 blocks; the minimum write size was 1 block, and the maximum write size was 128 blocks. The standard deviation was 53.5. The average write response time in the sample was 16.386 milliseconds; the minimum response time was 0.005 milliseconds, the maximum response time was 59.821 milliseconds. The standard deviation was 14.104. The number of 512 blocks written in sequence (write sequences) was 228. The length of the write sequence in units of 512 bytes was 144.2; the minimum 1, and the maximum 768 512 byte blocks. The standard deviation was 191.2. The number of write sequences was 228. The length of the average write sequences (shown as write seq.) was 144.2, a minimum of 1, and a maximum of 768 512 byte blocks. The standard deviation was 191.2. The number of seeks was 228 meaning that for every write sequence, a seek was done. The seek distance (seek dist) in 512 blocks was 5289064. The time to the next request averaged 20.418 milliseconds (minimum 0.009 milliseconds and maximum 2843.826 milliseconds). It tells you the rate at which the volume is being accessed; so, on average, every 20.418 milliseconds, a request was issued. The volume through-put in this measurement was 1441.4 KB per second, which gives a utilization of 40 percent, shown as 0.40.

Appendix A. Mirroring of the rootvg

These procedures only apply to AIX Version 4.2.1 and Version 4.3. With AIX Version 4.3.3, the remaining restrictions of the mirroring the rootvg white paper have been completely removed.

A.1 Function

Mirroring rootvg is a function that has been requested for a long time. Here are the two main reasons to implement the mirroring of rootvg:

- To provide the continuous operation of the AIX operating system in the event that a disk that is part of the operating system experiences failure, and there exists an active mirrored copy of the operating system on some other disk.
- To provide the ability to boot more than one disk of the rootvg in the event that another boot disk has failed. In some cases, the ability to boot from an alternate disk may require some user intervention.

A.2 Procedure

The following steps assume that the user has a rootvg that contains hdisk0, and the user is attempting to mirror the rootvg to a new disk hdisk1.

1. Extend rootvg to hdisk1:

```
extendvg rootvg hdisk1
```

If the user encounters the error message:

```
0516-050 Not enough descriptor space left in this volume group, Either try
adding a smaller PV or use another volume group.
```

the user, in this case, will not be allowed not add hdisk1 to the rootvg volume group.

The user has the following options:

- The user may attempt to mirror the rootvg volume group to another empty disk already in the rootvg.
- The user may attempt to use a smaller disk.

- The user may attempt to change the number of partitions supported per physical volume via `chvg -t`. (available on AIX 4.3.1 and later).

Note

Disks where the boot logical volume reside must be supported by AIX as boot devices. Check this by running the following command:

```
bootinfo -B hdiskX
```

If the command returns a "1", then it will be bootable by AIX. Any other value indicates that this disk is not a candidate for rootvg mirroring.

4. Mirror the rootvg by using the `mirrorvg` command with the exact mapping option:

```
mirrorvg -m rootvg hdisk1
```

Mirrorvg will turn off quorum if the volume group is rootrvg.

Note

If you do not use the exact mapping option of `mirrorvg`, then you must verify that the new copy of boot logical volume, hd5, is made of contiguous partitions.

5. Bosboot to initialize all boot records and devices: `bosboot -a`
6. Initialize the boot list: `bootlist -m normal hdisk0 hdisk1`

Note

Even though this command identifies the list of possible boot disks, it does not guarantee that the system will boot from the alternate disk in all cases involving failures of the first disk. In such situations, it may be necessary for the user to boot from the installation/maintenance media, select **maintenance**, reissue the `bootlist` command leaving out the failing disk, and then reboot. On some models, firmware provides a utility for electing the boot device at boot time. This may also be used to force the system to boot from the alternate disk.

Some hardware models do not support the `bootlist` command. This does not prevent the rootvg from being mirrored. This will require user intervention to select an alternate boot disk if the original disk is unavailable at the next reboot.

7. Shut down and reboot the system: `shutdown -Fr`

This is so that the QUORUM OFF functionality takes effect.

Appendix B. Sample scripts for HACMP configurations

This appendix provides some of the scripts discussed in the concuurent volume group chapter. You may want to refer to them during the lecture of this chapter.

B.1 Define-Cluster-Topology

```
#!/bin/ksh -x
#
# Define-Cluster-Topology -- used in Chap.11 of HACMP installation.
#
# See: SC23-4278-01 High Availabitily Cluster Multi-Processing for AIX
#                   Installation Guide Version 4.3.1
#
#                   Chapter.11 Defining the Cluster Topology.
#

################################################################
#
# Step.0: define PATH,LANG,ODMDIR environment values and export it.
#
################################################################

USER_PATH=/admin/bin
SYS_PATH=/usr/bin:/usr/bin/X11:/usr/ucb
ADMIN_PATH=/usr/sbin:/sbin:/etc
HACMP_ES_DIR=/usr/es/sbin/cluster
HACMP_ES_PATH=${HACMP_ES_DIR}:${HACMP_ES_DIR}/utilities:${HACMP_ES_DIR}/diag:${HACMP_ES_
DIR}/events:${HACMP_ES_DIR}/events/utils
HACMP_DIR=/usr/sbin/cluster
HACMP_PATH=${HACMP_DIR}:${HACMP_DIR}/utilities:${HACMP_DIR}/diag:${HACMP_DIR}/events:${H
ACMP_DIR}/events/utils
PSSP_PATH=/usr/lpp/ssp/bin:/usr/lpp/ssp/kerberos/bin:/usr/lpp/ssp/rcmd/bin
PATH=${HACMP_ES_PATH}:${HACMP_PATH}:${USER_PATH}:${PSSP_PATH}:${SYS_PATH}:${ADMIN_PATH}:
.

LANG=en_US

ODMDIR=/etc/objrepos

export PATH
export LANG
export ODMDIR

################################################################
#
# Step.1: Check this host is SP nodes or not.
#
################################################################

#SSP_INSTALL=”/usr/lpp/ssp/install/bin”
#
#${SSP_INSTALL}/node_number > /dev/null 2>&1
#RC=$?
#
#if [ ${RC} -ne 0 ]; then
#    echo “This script works on SP nodes only.”
#    exit 1
#fi
#
#NODE_NUMBER=‘$SSP_INSTALL/node_number‘

#case ${NODE_NUMBER} in
#1|3)
#        # do following steps.
#        ;;
#*)
#        # wrong node.
#        print -u2 “This script runs on node#1,3 only.”
#        exit 1
#        ;;
#esac

################################################################
#
# Step.2:
# Defining the Cluster ID and Name.
#
################################################################
claddclstr -i 1  -n disney
#             ^        ^
#             |        |
#             |   cluster name.
#         cluster ID.
RC=$?
echo “return value from claddclstr command is $RC.”

################################################################
#
# Step.3:
# Defining Nodes.
#
################################################################
clnodename -a mickey
RC=$?
echo “return value from clnodename command is $RC.”
clnodename -a goofy
RC=$?
echo “return value from clnodename command is $RC.”

################################################################
#
# Step.4:
# Defining Adapters.
#
################################################################
#
# Step.4.1: node#1 (mickey)
#

# ITSC_TRN (service).
claddnode -a mickey : token : TRN_ITSC : public : service : \
        9.3.187.183 : -n mickey
RC=$?
echo “return value from claddnode command is $RC.”
# ITSC_TRN (boot).
claddnode -a mickey_boot : token : TRN_ITSC : public : boot : \
        9.3.187.184 : -n mickey
RC=$?
echo “return value from claddnode command is $RC.”
```

```
# ITSC_TRN (stby).
claddnode -a mickey_stby : token : TRN_ITSC : public : standby : \
        9.3.187.251 : -n mickey
RC=$?
echo “return value from claddnode command is $RC.”
# ITSC_ETHER (service).
# Note: this network used for KA packet only.
claddnode -a mickey_en0 : ether : ETHER_ITSC : public : service : \
        100.100.100.1 : -n mickey
RC=$?
echo “return value from claddnode command is $RC.”
# ITSC_RS232 (service).
# Note: this network used for KA packet only.
claddnode -a mickey_tty : rs232 : RS232_ITSC : serial : service : \
        /dev/tty0 : -n mickey
RC=$?
echo “return value from claddnode command is $RC.”
#
# Step.4.2: node#2 (goofy)
#

# ITSC_TRN (service).
claddnode -a goofy : token : TRN_ITSC : public : service : \
        9.3.187.185 : -n goofy
RC=$?
echo “return value from claddnode command is $RC.”
# ITSC_TRN (boot).
claddnode -a goofy_boot : token : TRN_ITSC : public : boot : \
        9.3.187.186 : -n goofy
RC=$?
echo “return value from claddnode command is $RC.”
# ITSC_TRN (stby).
claddnode -a goofy_stby : token : TRN_ITSC : public : standby : \
        9.3.187.252 : -n goofy
RC=$?
echo “return value from claddnode command is $RC.”
# ITSC_ETHER (service).
# Note: this network used for KA packet only.
claddnode -a goofy_en0 : ether : ETHER_ITSC : public : service : \
        100.100.100.2 : -n goofy
RC=$?
echo “return value from claddnode command is $RC.”
# ITSC_RS232 (service).
# Note: this network used for KA packet only.
claddnode -a goofy_tty : rs232 : RS232_ITSC : serial : service : \
        /dev/tty0 : -n goofy
RC=$?
echo “return value from claddnode command is $RC.”

################################################################
#
# Step.6: Setting All nodes to Enhanced Security Mode
#
# !!! Note !!!
#
# In standard security mode, this step should be done in after the
# “Configuring-Cluster-Resources” shell script issued.
# If you use the enhanced security mode on SP systems, then the
# default security mode value “standard” will cause the failure
# of this script in step.7 with error code 2504-002.
# In that case, we set security mode to “Enhanced” here explicitly.
#
################################################################
```

```
#clchparam -n sp01e VERBOSE_LOGGING=high\
#NAME_SERVER=false
#RC=$?
#echo “return value from clchparam command is $RC.”
#clchparam -n sp03e VERBOSE_LOGGING=high\
#NAME_SERVER=false
#RC=$?

#clchclstr -s Enhanced
#RC=$?
#echo “return value from clchparam command is $RC.”

################################################################
#
# Step.7: Synchronize the Cluster Definition Across Nodes
#
# !!! Note !!!
#
# 1. This step should be done manually on one node in cluster.
#
# 2. Actually, this step is done via SMIT with “cldare” command.
# We define cluster topology in cold method (with out dynamic
# resource definition), so DARE function does not needed.
#
################################################################
#clconfig -s -t
#RC=$?
#echo “return value from clconfig command is $RC.”
echo “you have to syncronize the cluster topology now.”
echo “issue \”clconfig -s -t\” on only one node.”

#
# end of Define-Cluster-Topology
#
```

B.2 Configuring-Cluster-Resources

```
#!/bin/ksh -x
#
# Configuring-Cluster-Resources -- used in Chap.12 of HACMP installation.
#
# See: SC23-4278-01 High Availabitily Cluster Multi-Processing for AIX
#                   Installation Guide Version 4.3.1
#
#                   Chapter.12 Configuring Cluster Resources.
#

################################################################
#
# Step.0: define PATH,LANG,ODMDIR environment values and export it.
#
################################################################

USER_PATH=/admin/bin
SYS_PATH=/usr/bin:/usr/bin/X11:/usr/ucb
ADMIN_PATH=/usr/sbin:/sbin:/etc
HACMP_DIR=/usr/sbin/cluster
HACMP_PATH=${HACMP_DIR}:${HACMP_DIR}/utilities:${HACMP_DIR}/diag:${HACMP_DIR}/events:${H
ACMP_DIR}/events/utils
PSSP_PATH=/usr/lpp/ssp/bin:/usr/lpp/ssp/kerberos/bin:/usr/lpp/ssp/rcmd/bin
PATH=${HACMP_PATH}:${USER_PATH}:${PSSP_PATH}:${SYS_PATH}:${ADMIN_PATH}:.
```

```
LANG=en_US

ODMDIR=/etc/objrepos

export PATH
export LANG
export ODMDIR
export CONFIG_PATH

################################################################
#
# Step.1: Check this host is SP nodes or not.
#
################################################################

#SSP_INSTALL=”/usr/lpp/ssp/install/bin”
#
#${SSP_INSTALL}/node_number > /dev/null 2>&1
#RC=$?
#
#if [ ${RC} -ne 0 ]; then
#    echo “This script works on SP nodes only.”
#    exit 1
#fi
#
#NODE_NUMBER=‘$SSP_INSTALL/node_number‘

#case ${NODE_NUMBER} in
#1|3)
#        # do following steps.
#        ;;
#*)
#        # wrong node.
#        print -u2 “This script runs on node#1,3 only.”
#        exit 1
#        ;;
#esac

################################################################
#
# Step.2:
# Creating Resource Groups
#
################################################################
#
# Step.2.1: node#1, 2 concurrent resource group.
#
claddgrp -g conc_rg -r concurrent -n ‘mickey goofy’
RC=$?
echo “return value from claddgrp command is $RC.”

#
# Step.2.2: node#1 -> 2 cascading resource group.
#
claddgrp -g 1_2_rg -r cascading -n ‘mickey goofy’
RC=$?
echo “return value from claddgrp command is $RC.”

#
# Step.2.3: node#2 -> 1 cascading resource group.
#
claddgrp -g 2_1_rg -r cascading -n ‘goofy mickey’
```

```
RC=$?
echo “return value from claddgrp command is $RC.”

################################################################
#
# Step.3:
# Configuring Application Servers
#
################################################################
#
# Step.3.1: development server (node#01 -> 05).
#
#claddserv -s develop_1_5.server \
#    -b “/admin/HACMP/APPL/develop_1_5.server.start”\
#    -e “/admin/HACMP/APPL/develop_1_5.server.stop”
#RC=$?
#echo “return value from claddserv command is $RC.”
################################################################
#
# Step.4:
# Configuring Resources for Resource Groups
#
################################################################
#
# Step.4.1: define node#1, 2 concurrent resource group.
#
claddres -g ‘conc_rg’\
        SERVICE_LABEL=\
        ILESYSTEM=\
        FSCHECK_TOOL=’fsck’\
        RECOVERY_METHOD=’sequential’\
        EXPORT_FILESYSTEM=\
        MOUNT_FILESYSTEM=\
        VOLUME_GROUP=\
        CONCURRENT_VOLUME_GROUP=’concvg’\
        DISK=\
        AIX_CONNECTIONS_SERVICES=\
        AIX_FAST_CONNECT_SERVICES=\
        APPLICATIONS=\
        SNA_CONNECTIONS=\
        MISC_DATA=\
        INACTIVE_TAKEOVER=’false’\
        DISK_FENCING=’false’\
        SSA_DISK_FENCING=’false’\
        FS_BEFORE_IPADDR=’false’
RC=$?
echo “return value from claddres command is $RC.”

#
# Step.4.2: define node#1 -> 2 cascading resource group.
#
claddres -g ‘1_2_rg’\
        SERVICE_LABEL=’mickey’
        FILESYSTEM=\
        FSCHECK_TOOL=’fsck’\
        RECOVERY_METHOD=’sequential’\
        EXPORT_FILESYSTEM=\
        MOUNT_FILESYSTEM=\
        VOLUME_GROUP=\
        CONCURRENT_VOLUME_GROUP=\
        DISK=\
        AIX_CONNECTIONS_SERVICES=\
        AIX_FAST_CONNECT_SERVICES=\
        APPLICATIONS=\
        SNA_CONNECTIONS=\
        MISC_DATA=\
        INACTIVE_TAKEOVER=’false’\
        DISK_FENCING=’false’\
        SSA_DISK_FENCING=’false’\
        FS_BEFORE_IPADDR=’false’
RC=$?
echo “return value from claddres command is $RC.”

#
# Step.4.3: define node#2 -> 1 cascading resource group.
#
claddres -g ‘2_1_rg’\
        SERVICE_LABEL=’goofy’
        FILESYSTEM=\
        FSCHECK_TOOL=’fsck’\
        RECOVERY_METHOD=’sequential’\
        EXPORT_FILESYSTEM=\
        MOUNT_FILESYSTEM=\
        VOLUME_GROUP=\
        CONCURRENT_VOLUME_GROUP=\
        DISK=\
        AIX_CONNECTIONS_SERVICES=\
        AIX_FAST_CONNECT_SERVICES=\
        APPLICATIONS=\
        SNA_CONNECTIONS=\
        MISC_DATA=\
        INACTIVE_TAKEOVER=’false’\
        DISK_FENCING=’false’\
        SSA_DISK_FENCING=’false’\
        FS_BEFORE_IPADDR=’false’
RC=$?
echo “return value from claddres command is $RC.”
################################################################
#
# Step.5:
# Configuring Run-Time Parameters
#
# !!! Note !!!
# This step should be done in Defing-Cluster-Topology, if you use
# the enhanced security mode in the cluster (used in SP systems only).
################################################################
clchparam -n mickey VERBOSE_LOGGING=high\
       NAME_SERVER=true
RC=$?
echo “return value from clchparam command is $RC.”
clchparam -n goofy VERBOSE_LOGGING=high\
       NAME_SERVER=true
RC=$?


################################################################
# Step.9:
# Synchronizing Cluster Nodes
# Note: After this, modificatin to /etc/inittab and /etc/rc.net is permanent.
#       These modification is documented on SC23-4278-01,
#       Appendix.A Supporting IP address Takeover.
################################################################
clconfig -s -r
RC=$?
echo “return value from clchparm command is $RC.”
```

```
#
# end of Configuring-Cluster-Resources
#
```

B.3 Unconfigure-Cluster

```
#!/bin/ksh -x
#
# Unconfigure-Cluster -- Unconfigure HACMP cluster configuration.
#
# See: SC23-4278-00 High Availabitily Cluster Multi-Processing for AIX
#                   Installation Guide Version 4.3
#

################################################################
#
# Step.0: define PATH,LANG,ODMDIR environment values and export it.
#
################################################################

USER_PATH=/admin/bin
SYS_PATH=/usr/bin:/usr/bin/X11:/usr/ucb
ADMIN_PATH=/usr/sbin:/sbin:/etc
HACMP_DIR=/usr/sbin/cluster
HACMP_PATH=${HACMP_DIR}:${HACMP_DIR}/utilities:${HACMP_DIR}/diag:${HACMP_DIR}/events:${H
ACMP_DIR}/events/utils
PSSP_PATH=/usr/lpp/ssp/bin:/usr/lpp/ssp/kerberos/bin:/usr/lpp/ssp/rcmd/bin
PATH=${HACMP_PATH}:${USER_PATH}:${PSSP_PATH}:${SYS_PATH}:${ADMIN_PATH}:.

LANG=en_US

ODMDIR=/etc/objrepos

export PATH
export LANG
export ODMDIR
export CONFIG_PATH

################################################################
#
# Step.1: Check this node has HACMP cluster configuration or not.
#         If have it, remove cluster configuration using “clrmclstr”.
#
################################################################


cllsclstr > /dev/null 2>&1
RC=$?

if [ ${RC} -ne 0 ]; then
    print -u2 “This node does not have HACMP cluster configuration.”
    exit 1
else
    clrmclstr
fi

################################################################
#
# Step.2: Check this node has HACMP application server or not.
#         If have it, remove all application server using “clrmserv”.
#
```

```
################################################################

cllsserv > /dev/null 2>&1
RC=$?

if [ ${RC} -ne 0 ]; then
    print -u2 “This node does not have HACMP cluster configuration.”
else
    clrmserv All
fi

################################################################
#
# Step.3: remove topology service stanza from HACMPtopsvcs class.
#
# Note: used in HACMP/ES cluster only.
#
################################################################

#ODMDIR=/etc/objrepos odmdelete -o HACMPtopsvcs -q hbInterval=1

#
# end of Unconfigure-Cluster
#
```

Appendix C. /usr/sbin/cluster/hc/README

The HC daemon provided with HACMP 4.1.1 is essentially the same as that provided as part of the IBM Recoverable Virtual Shared Disk Licensed Program Product (program number 5765-444) described in *IBM Recoverable Virtual Shared Disk User's Guide and Reference*, GC23-3849. The only changes were those necessary to make it function in an HACMP as opposed to an RVSD environment. The description below is based on the content of the relevant section of the redbook *IBM Recoverable Virtual Shared Disk User's Guide and Reference*, GC23-3849. The HC daemon and supporting utilities and shell scripts are installed in the directory /usr/lpp/csd/bin. HACMP 4.1.1 will not install this support if the RVSD is already present in the system. If HACMP 4.1.1 is installed, and the RVSD is to be installed later, the cluster.hc.rte fileset should be removed to prevent possible incompatibilites.

You can code your program to use the HC application programming interface of the IBM Recoverable Virtual Shared Disk software to aid in application recovery.

Your program should include the following hc.h header file, which is installed in the /usr/sbin/cluster/hc directory:

```
#include "hc.c"
```

Your program connects to a socket where HC is listening. If a node fails, or the client application on a node fails, HC sends a reconfiguration message according to the protocol defined in the hc.h header file. Your program should check for messages from that socket and react accordingly. Note, that only one application can use HC's services. This application should be the only application using HC on any node.

The HC daemon was originally developed to support the Oracle Fault Tolerant Distributed Lock manager. Consult Oracle documentation for the appropriate release of Oracle Parallel Server to make use of this facility in HACMP 4.1.1.

As a convenience, the HC program will run an optional script:

```
/usr/lpp/csd/bin/hc.activate
```

once it initializes. The hc.activate script is passed two parameters: The local node number and the path name of the UNIX domain stream socket that it serves. You can start your recoverable application from this script. (Node numbers correspond to the sequence in which nodes are given in the output of the /usr/sbin/cluster/utilities/cllsif utility.)

The HC program also runs an optional script:

```
/usr/lpp/csd/bin/hc.deactivate
```

upon detecting failure of its client. The same parameters are passed as are passed to /usr/lpp/csd/bin/hc.activate, letting you restart the client if you desire. The client may fail either by exiting or by failure to respond to a ping message within 10 seconds.

Note

Ten seconds is the default for the PING_DELAY. That is, if a client application fails to respond to a ping sent from the hc program within 10 seconds, HC will consider the application, or the node it is running on, to have failed. Some application programs running under some system loads may require a longer PING_DELAY time frame. To change the PING_DELAY, place a new value in /usr/lpp/csd/bin/hacmp_hc_start. For example, add the following line:

```
export PING_DELAY=30
```

If HC should fail, the application will receive a zero-length message over the socket and should shut itself down.

Appendix D. The filemon.out file

```
Mon Oct  4 15:19:46 1999
System: AIX lorraine Node: 4 Machine: 000416314C00
11.323 secs in measured interval
Cpu utilization:  24.6%

Most Active Files
------------------------------------------------------------------------
  #MBs  #opns   #rds   #wrs  file                 volume:inode
------------------------------------------------------------------------
  16.0      1  32769      0  hd9var
  16.0      1      0  32768  datafile.1           /dev/inner:17
   0.1     12     14      0  group                /dev/hd4:4110
   0.0      1     11      0  cmdnfs.cat           /dev/hd2:41871
   0.0      1      6      0  user                 /dev/hd4:39
   0.0      1      6      0  limits               /dev/hd4:30
   0.0      1      6      0  environ              /dev/hd4:26
   0.0      8      7      0  passwd               /dev/hd4:4145
   0.0      1      4      0  config               /dev/hd4:699
   0.0      2      4      0  ksh.cat              /dev/hd2:41602
   0.0      1      3      0  methods.cfg          /dev/hd2:6300
   0.0      1      2      0  cmdtrace.cat         /dev/hd2:41470
   0.0      8      2      0  passwd               /dev/hd4:32
   0.0      1      2      0  environment          /dev/hd4:4108
   0.0      1      2      0  resolv.conf          /dev/hd4:4346
   0.0      1      1      0  NIS_COLD_START       /dev/hd9var:2234
   0.0      1      1      0  dd.cat               /dev/hd2:41493
   0.0      1      0      1  active_list.tmp
   0.0      1      0      1  active_file.rpc.nispasswdd.18878
   0.0      1      0      1  error

Most Active Segments
------------------------------------------------------------------------
  #MBs  #rpgs  #wpgs  segid  segtype              volume:inode
------------------------------------------------------------------------
  16.0      0   4092   3f66  page table
   0.0      0      6   b877  log
   0.0      0      2   41e8  page table
   0.0      0      1   86f1  page table
   0.0      0      1   af54  log

Most Active Logical Volumes
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.52      0  32744 1445.9  /dev/inner           /inner
  0.51  32776     16 1448.0  /dev/hd9var          /var Frag_Sz.= 512
  0.01      0     48    2.1  /dev/hd8             jfslog
  0.00      0      8    0.4  /dev/loglv00         jfslog

Most Active Physical Volumes
```

```
------------------------------------------------------------------------
  util  #rblk  #wblk   KB/s  volume               description
------------------------------------------------------------------------
  0.40      0  32867 1451.4  /dev/hdisk2          N/A
  0.40      0  32867 1451.4  /dev/hdisk1          N/A
  0.27  32768     64 1449.8  /dev/hdisk0          N/A
  0.00      0      8    0.4  /dev/hdisk3          N/A

------------------------------------------------------------------------
Detailed File Stats
------------------------------------------------------------------------

FILE: /dev/hd9var
opens:                  1
total bytes xfrd:       16777728
reads:                  32769   (0 errs)
  read sizes (bytes):   avg   512.0 min     512 max     512 sdev     0.0
read times (msec):    avg   0.112 min   0.006 max  30.543 sdev   0.604

FILE: /inner/datafile.1  volume: /dev/inner (/inner)  inode: 17
opens:                  1
total bytes xfrd:       16777216
writes:                 32768   (0 errs)
  write sizes (bytes):  avg   512.0 min     512 max     512 sdev     0.0
  write times (msec):   avg   0.029 min   0.008 max   1.459 sdev   0.048
lseeks:                 1

FILE: /etc/group  volume: /dev/hd4 (/)  inode: 4110
opens:                  12
total bytes xfrd:       57344
reads:                  14      (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.015 min   0.004 max   0.047 sdev   0.010

FILE: /usr/lib/nls/msg/en_US/cmdnfs.cat  volume: /dev/hd2 (/usr)  inode: 41871
opens:                  1
total bytes xfrd:       45056
reads:                  11      (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.040 min   0.023 max   0.092 sdev   0.019
lseeks:                 9

FILE: /etc/security/user  volume: /dev/hd4 (/)  inode: 39
opens:                  1
total bytes xfrd:       41480
reads:                  6       (0 errs)
  read sizes (bytes):   avg  6913.3 min      76 max   16383 sdev  6877.7
  read times (msec):    avg   0.030 min   0.005 max   0.124 sdev   0.043
lseeks:                 10

FILE: /etc/security/limits  volume: /dev/hd4 (/)  inode: 30
opens:                  1
total bytes xfrd:       41144
reads:                  6       (0 errs)
```

```
  read sizes (bytes):   avg  6857.3 min      81 max   16383 sdev  6931.3
  read times (msec):    avg   0.010 min   0.005 max   0.029 sdev   0.009
lseeks:                 10

FILE: /etc/security/environ  volume: /dev/hd4 (/)  inode: 26
opens:                  1
total bytes xfrd:       40962
reads:                  6       (0 errs)
  read sizes (bytes):   avg  6827.0 min       1 max   16383 sdev  6960.9
  read times (msec):    avg   0.007 min   0.005 max   0.014 sdev   0.003
lseeks:                 10

FILE: /etc/passwd  volume: /dev/hd4 (/)  inode: 4145
opens:                  8
total bytes xfrd:       28672
reads:                  7       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.031 min   0.016 max   0.052 sdev   0.013
lseeks:                 10

FILE: /etc/security/audit/config  volume: /dev/hd4 (/)  inode: 699
opens:                  1
total bytes xfrd:       24594
reads:                  4       (0 errs)
  read sizes (bytes):   avg  6148.5 min      17 max   16383 sdev  6139.2
  read times (msec):    avg   0.011 min   0.003 max   0.032 sdev   0.012
lseeks:                 5

FILE: /usr/lib/nls/msg/en_US/ksh.cat  volume: /dev/hd2 (/usr)  inode: 41602
opens:                  2
total bytes xfrd:       16384
reads:                  4       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.052 min   0.015 max   0.088 sdev   0.036
lseeks:                 10

FILE: /usr/lib/security/methods.cfg  volume: /dev/hd2 (/usr)  inode: 6300
opens:                  1
total bytes xfrd:       12288
reads:                  3       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.022 min   0.003 max   0.055 sdev   0.023
lseeks:                 54

FILE: /usr/lib/nls/msg/en_US/cmdtrace.cat  volume: /dev/hd2 (/usr)  inode: 41470
opens:                  1
total bytes xfrd:       8192
reads:                  2       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.063 min   0.044 max   0.081 sdev   0.019
lseeks:                 8

FILE: /etc/security/passwd  volume: /dev/hd4 (/)  inode: 32
opens:                  8
```

```
total bytes xfrd:       8192
reads:                  2       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.015 min   0.015 max   0.016 sdev   0.001
lseeks:                 12

FILE: /etc/environment  volume: /dev/hd4 (/)  inode: 4108
opens:                  1
total bytes xfrd:       8192
reads:                  2       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.021 min   0.004 max   0.039 sdev   0.017

FILE: /etc/resolv.conf  volume: /dev/hd4 (/)  inode: 4346
opens:                  1
total bytes xfrd:       8192
reads:                  2       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.023 min   0.004 max   0.041 sdev   0.019

FILE: /var/nis/NIS_COLD_START  volume: /dev/hd9var (/var)  inode: 2234
opens:                  1
total bytes xfrd:       4096
reads:                  1       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.047 min   0.047 max   0.047 sdev   0.000

FILE: /usr/lib/nls/msg/en_US/dd.cat  volume: /dev/hd2 (/usr)  inode: 41493
opens:                  1
total bytes xfrd:       4096
reads:                  1       (0 errs)
  read sizes (bytes):   avg  4096.0 min    4096 max    4096 sdev     0.0
  read times (msec):    avg   0.113 min   0.113 max   0.113 sdev   0.000
lseeks:                 5

FILE: /var/adm/SRC/active_list.tmp
opens:                  1
total bytes xfrd:       2041
writes:                 1       (0 errs)
  write sizes (bytes):  avg  2041.0 min    2041 max    2041 sdev     0.0
  write times (msec):   avg   0.163 min   0.163 max   0.163 sdev   0.000

FILE: /var/adm/SRC/active_file.rpc.nispasswdd.18878
opens:                  1
total bytes xfrd:       173
writes:                 1       (0 errs)
  write sizes (bytes):  avg   173.0 min     173 max     173 sdev     0.0
  write times (msec):   avg   0.109 min   0.109 max   0.109 sdev   0.000

FILE: /dev/error
opens:                  1
total bytes xfrd:       92
writes:                 1       (0 errs)
  write sizes (bytes):  avg    92.0 min      92 max      92 sdev     0.0
  write times (msec):   avg   5.672 min   5.672 max   5.672 sdev   0.000

------------------------------------------------------------------------
Detailed VM Segment Stats   (4096 byte pages)
------------------------------------------------------------------------

SEGMENT: 3f66  segtype: page table
segment flags:          pgtbl
writes:                 4092    (0 errs)
  write times (msec):   avg  74.073 min   6.809 max 172.287 sdev  35.678
  write sequences:      29
  write seq. lengths:   avg   141.1 min       1 max    4064 sdev   741.4

SEGMENT: b877  segtype: log
segment flags:          log
writes:                 6       (0 errs)
  write times (msec):   avg  16.460 min   7.008 max  28.417 sdev   6.266
  write sequences:      5
  write seq. lengths:   avg     1.2 min       1 max       2 sdev     0.4

SEGMENT: 41e8  segtype: page table
segment flags:          sys pgtbl
writes:                 2       (0 errs)
  write times (msec):   avg  15.046 min  10.745 max  19.347 sdev   4.301
  write sequences:      2
  write seq. lengths:   avg     1.0 min       1 max       1 sdev     0.0

SEGMENT: 86f1  segtype: page table
segment flags:          pers log pgtbl
writes:                 1       (0 errs)
  write times (msec):   avg  17.895 min  17.895 max  17.895 sdev   0.000
  write sequences:      1
  write seq. lengths:   avg     1.0 min       1 max       1 sdev     0.0

SEGMENT: af54  segtype: log
segment flags:          log
writes:                 1       (0 errs)
  write times (msec):   avg   8.203 min   8.203 max   8.203 sdev   0.000
  write sequences:      1
  write seq. lengths:   avg     1.0 min       1 max       1 sdev     0.0

------------------------------------------------------------------------
Detailed Logical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/inner  description: /inner
writes:                 1118    (0 errs)
  write sizes (blks):   avg    29.3 min       8 max      32 sdev     6.6
  write times (msec):   avg  74.969 min   6.785 max 172.230 sdev  35.819
  write sequences:      9
  write seq. lengths:   avg  3638.2 min       8 max    8192 sdev  3189.3
seeks:                  9       (0.8%)
  seek dist (blks):     avg  1164.0 min       8 max    8216 sdev  2674.3
time to next req(msec): avg   7.877 min   0.007 max 2843.664 sdev  84.943
```

```
throughput:             1445.9 KB/sec
utilization:            0.52

VOLUME: /dev/hd9var  description: /var Frag_Sz.= 512
reads:                  4097    (0 errs)
  read sizes (blks):    avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):    avg   0.793 min   0.363 max  30.601 sdev   1.561
  read sequences:       2
  read seq. lengths:    avg 16388.0 min    9736 max   23040 sdev  6652.0
writes:                 2       (0 errs)
  write sizes (blks):   avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):   avg  15.011 min  10.704 max  19.318 sdev   4.307
  write sequences:      2
  write seq. lengths:   avg     8.0 min       8 max       8 sdev     0.0
seeks:                  3       (0.1%)
  seek dist (blks):     avg  8384.7 min    4785 max   12569 sdev  3204.6
time to next req(msec): avg   2.149 min   0.012 max 2876.785 sdev  44.934
throughput:             1448.0 KB/sec
utilization:            0.51

VOLUME: /dev/hd8  description: jfslog
writes:                 6       (0 errs)
  write sizes (blks):   avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):   avg  16.411 min   6.846 max  28.393 sdev   6.301
  write sequences:      4
  write seq. lengths:   avg    12.0 min       8 max      16 sdev     4.0
seeks:                  4       (66.7%)
  seek dist (blks):     avg    16.0 min       8 max      24 sdev     6.5
time to next req(msec): avg 792.633 min   6.876 max 4652.208 sdev 1726.078
throughput:             2.1 KB/sec
utilization:            0.01

VOLUME: /dev/loglv00  description: jfslog
writes:                 1       (0 errs)
  write sizes (blks):   avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):   avg   8.193 min   8.193 max   8.193 sdev   0.000
  write sequences:      1
  write seq. lengths:   avg     8.0 min       8 max       8 sdev     0.0
seeks:                  1       (100.0%)
  seek dist (blks):     init    160
time to next req(msec): avg 2861.933 min 2861.933 max 2861.933 sdev   0.000
throughput:             0.4 KB/sec
utilization:            0.00

------------------------------------------------------------------------
Detailed Physical Volume Stats   (512 byte blocks)
------------------------------------------------------------------------

VOLUME: /dev/hdisk2  description: N/A
writes:                 511     (0 errs)
  write sizes (blks):   avg    64.3 min       1 max     128 sdev    53.5
  write times (msec):   avg  16.386 min   0.005 max  59.821 sdev  14.104
  write sequences:      228
  write seq. lengths:   avg   144.2 min       1 max     768 sdev   191.2
```

```
seeks:                  228     (44.6%)
  seek dist (blks):     avg 4573452.9 min       1 max 5314558 sdev 1818766.1
  seek dist (%tot blks):avg 51.88927 min 0.00001 max 60.29767 sdev 20.63527
time to next req(msec): avg  20.418 min   0.009 max 2843.826 sdev 137.186
throughput:             1451.4 KB/sec
utilization:            0.40

VOLUME: /dev/hdisk1  description: N/A
writes:                 515     (0 errs)
  write sizes (blks):   avg    63.8 min       1 max     128 sdev    53.3
  write times (msec):   avg  15.540 min   0.005 max  70.753 sdev  14.057
  write sequences:      229
  write seq. lengths:   avg   143.5 min       1 max     768 sdev   190.3
seeks:                  229     (44.5%)
  seek dist (blks):     avg 4600014.3 min       1 max 5314558 sdev 1790467.7
  seek dist (%tot blks):avg 52.19063 min 0.00001 max 60.29767 sdev 20.31421
time to next req(msec): avg  20.417 min   0.012 max 2843.725 sdev 137.179
throughput:             1451.4 KB/sec
utilization:            0.40

VOLUME: /dev/hdisk0  description: N/A
reads:                  4096    (0 errs)
  read sizes (blks):    avg     8.0 min       8 max       8 sdev     0.0
  read times (msec):    avg   0.754 min   0.050 max  65.644 sdev   1.748
  read sequences:       8
  read seq. lengths:    avg  4096.0 min       8 max   22936 sdev  7799.3
writes:                 8       (0 errs)
  write sizes (blks):   avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):   avg  15.927 min   0.185 max  62.117 sdev  19.553
  write sequences:      8
  write seq. lengths:   avg     8.0 min       8 max       8 sdev     0.0
seeks:                  16      (0.4%)
seek dist (blks):     avg 127079.6 min    4769 max  156792 sdev 59367.5
  seek dist (%tot blks):avg 0.71497 min 0.02683 max 0.88213 sdev 0.33401
time to next req(msec): avg   2.147 min   0.164 max 2876.820 sdev  44.910
throughput:             1449.8 KB/sec
utilization:            0.27

VOLUME: /dev/hdisk3  description: N/A
writes:                 1       (0 errs)
  write sizes (blks):   avg     8.0 min       8 max       8 sdev     0.0
  write times (msec):   avg   8.155 min   8.155 max   8.155 sdev   0.000
  write sequences:      1
  write seq. lengths:   avg     8.0 min       8 max       8 sdev     0.0
seeks:                  1       (100.0%)
  seek dist (blks):     init 3576224
  seek dist (%tot blks):init 20.12035
time to next req(msec): avg 2861.948 min 2861.948 max 2861.948 sdev   0.000
throughput:             0.4 KB/sec
utilization:            0.00
```

Appendix E. Filemon syntax

`Filemon` monitors the performance of the file system and reports the I/O activity on behalf of logical files, virtual memory segments, logical volumes, and physical volumes.

Syntax

```
filemon [ -d  ] [ -i File ] [ -o File ] [ -O Levels ] [ -P  ] [ -T n ] [ -u
] [ -v  ]
```

Description

The `filemon` command monitors a trace of file system and I/O system events and reports on the file and I/O access performance during that period.

In its normal mode, the `filemon` command runs in the background, while one or more application programs or system commands are being executed and monitored. The `filemon` command automatically starts and monitors a trace of the program's file system and I/O events in real time. By default, the trace is started immediately; optionally, tracing may be deferred until the user issues a `trcon` command. The user can issue `trcoff` and `trcon` commands while the `filemon` command is running in order to turn off and on monitoring as desired. When tracing is stopped by a `trcstop` command, the `filemon` command generates an I/O activity report and exits.

The `filemon` command can also process a trace file that has been previously recorded by the AIX trace facility. The file and I/O activity report will be based on the events recorded in that file.

To provide a more complete understanding of file system performance for an application, the `filemon` command monitors file and I/O activity at four levels.

Logical file system

The `filemon` command monitors logical I/O operations on logical files. The monitored operations include all `read`, `write`, `open`, and `lseek` system calls, which may or may not result in actual physical I/O, depending on whether or not the files are already buffered in memory. I/O statistics are kept on a per-file basis.

Virtual memory system

The `filemon` command monitors physical I/O operations (that is, paging) between segments and their images on disk. I/O statistics are kept on a per-segment basis.

Logical volumes

The filemon command monitors I/O operations on logical volumes. I/O statistics are kept on a per-logical-volume basis.

Physical volumes

The filemon command monitors I/O operations on physical volumes. At this level, physical resource utilizations are obtained. I/O statistics are kept on a per-physical-volume basis.

Any combination of the four levels can be monitored, as specified by the command line flags. By default, the filemon command only monitors I/O operations at the virtual memory, logical volume, and physical volume levels. These levels are all concerned with requests for real disk I/O.

The filemon command writes its report to standard output or to a specified file. The report begins with a summary of the I/O activity for each of the levels being monitored and ends with detailed I/O activity statistics for each of the levels being monitored. Summary and detailed report contents are described in the "Reports " section.

Note

The reports produced by the filemon command can be quite long. Consequently, the -o option should usually be used to write the report to an output file. When a physical device is opened and accessed directly by an application, only reads and writes of complete 512-byte blocks are reflected in the report. "Short" reads and writes, used by the device driver to issue device commands and read device status, are ignored. CD-ROMs do not have concentric "tracks" or "cylinders," as in hard files. (There is one spiral track.) Consequently, it is not possible to report seek distance statistics for CD-ROMs in terms of cylinders. The -u flag is used to generate reports on files opened prior to the start of the trace daemon. Some of this data can be useful, but much of it applies to daemons and other unrelated activity. This background information can be overwhelming, especially on large systems. If the /unix file and the running kernel are not the same, then the kernel addresses will be incorrect, thus, causing the filemon command to exit.

When using the filemon command from within a shell script, allow for a slight delay prior to viewing the contents of the filemon output file. The filemon command may take a few seconds to produce this report.

System Trace Facility

The `filemon` command obtains raw I/O performance data using the AIX system `trace` facility. Currently, the `trace` facility only supports one output stream. Consequently, only one `filemon` or `trace` process can be active at a time. If another `filemon` or `trace` process is already running, the `filemon` command will respond with the message:

```
/dev/systrace: Device busy
```

While monitoring very I/O-intensive applications, the `filemon` command may not be able to consume trace events as fast as they are produced in real time. When that happens, the error message:

```
Trace kernel buffers overflowed, N missed entries
```

will be displayed on `stderr`, indicating how many trace events were lost while the trace buffers were full. The `filemon` command will continue monitoring I/O activity, but the accuracy of the report will be diminished to some unknown degree. One way to prevent overflow is to monitor fewer levels of the file and I/O subsystems: The number of trace events generated is proportional to the number of levels monitored. Additionally, the trace buffer size can be increased using the `-T` option to accommodate larger bursts of trace events before overflow. Remember that increasing the trace buffer size will result in more pinned memory and, therefore, may effect I/O and paging behavior.

In memory-constrained environments (where demand for memory exceeds supply), the `-P` option can be used to pin the text and data pages of the real-time `filemon` process in memory so that the pages cannot be swapped out. If the `-P` option is not used, allowing the `filemon` process to be swapped out, the progress of the `filemon` command may be delayed to the point where it cannot process trace events fast enough. This situation leads to trace buffer overflow as described above. Of course, pinning this process takes memory away from the application (although the `filemon` command is not a large program; its process image can consume up to 500 KB).

Before using the `filemon` command to process an existing trace data file, you must use the `-r` option of the `trcrpt` command to rewrite the trace data sequentially to a new file. Otherwise, the `filemon` command produces the following error message and then exits:

```
error: run 'trcrpt -r' on logfile first
```

Reports

Each report generated by the `filemon` command has a header that identifies the date, the machine ID, and the length of the monitoring period in seconds. The CPU utilization during the monitoring period is also reported.

Next, summary reports are generated for each of the file system levels being monitored. By default, the logical file and virtual memory reports are limited to the 20 most active files and segments, respectively, as measured by the total amount of data transferred. If the `-v` flag has been specified, activity for all files and segments is reported. There is one row for each reported file, segment, or volume. The columns in each row for the four summary reports are described in the following lists:

Most Active Files Report

Column	Description
#MBS	Total number of megabytes transferred to/from file. The rows are sorted by this field in decreasing order.
#opns	Number of times the file was opened during measurement period.
#rds	Number of read system calls made against file
#wrs	Number of write system calls made against file.
file	Name of file (full path name is in detailed report).
volume:i-node	Name of volume that contains the file and the file's i-node number. This field can be used to associate a file with its corresponding persistent segment shown in the virtual memory I/O reports. This field may be blank for example, for temporary files created and deleted during execution.

Most Active Segments Report

Column	Description
#MBS	Total number of megabytes transferred to/from segment. The rows are sorted by this field in decreasing order.
#rpgs	Number of 4096-byte pages read into segment from disk (that is, page).

#wpgs Number of 4096-byte pages written from segment to disk (page out).

segid Internal ID of segment.

segtype Type of segment: Working segment, persistent segment (local file), client segment (remote file), page table segment, system segment, or special persistent segments containing file system data (log, root directory, .inode, .inodemap, .inodex, .inodexmap, .indirect, .diskmap).

volume:i-node For persistent segments, name of volume that contains the associated file and the file's i-node number. This field can be used to associate a persistent segment with its corresponding file shown in the file I/O reports. This field will be blank for non-persistent segments.

Note

The virtual memory analysis tool, `svmon`, can be used to display more information about a segment, given its segment ID (segid), as follows:

```
Svmon -S <segid>
```

Most Active Logical Volumes Report

Column Description

util Utilization of the volume (fraction of time busy). The rows are sorted by this field in decreasing order.

#rblk Number of 512-byte blocks read from the volume.

#wblk Number of 512-byte blocks written to the volume.

KB/sec Total transfer throughput in Kilobytes per second.

volume Name of volume.

description Contents of volume: Either a file system name or logical volume type (paging, jfslog, boot, or sysdump). Also, indicates if the file system is fragmented or compressed.

Most Active Physical Volumes Report

Column Description

util Utilization of the volume (fraction of time busy). The rows are sorted by this field in decreasing order.

#rblk Number of 512-byte blocks read from the volume.

#wblk Number of 512-byte blocks written to the volume.

KB/sec Total volume throughput in Kilobytes per second.

volume Name of volume.

description Type of volume, for example, 120 MB disk, 355 MB SCSI, or CDROM SCSI.

> **Note**
>
> Logical volume I/O requests start before, and end after, physical volume I/O requests. For this reason, total logical volume utilization will appear to be higher than total physical volume utilization.

Finally, detailed reports are generated for each of the file system levels being monitored. By default, the logical file and virtual memory reports are limited to the 20 most active files and segments, respectively, as measured by the total amount of data transferred. If the `-v` flag has been specified, activity for all files and segments is reported. There is one entry for each reported file, segment, or volume. The fields in each entry are described below for the four detailed reports are described in the following lists.

Some of the fields report a single value; others report statistics that characterize a distribution of many values. For example, response time statistics are kept for all read or write requests that were monitored. The average, minimum, and maximum response times are reported as well as the standard deviation of the response times. The standard deviation is used to show how much the individual response times deviated from the average. Roughly two-thirds of the sampled response times are between average - standard deviation and average + standard deviation. If the distribution of response times is scattered over a large range; the standard deviation will be large compared to the average response time.

Detailed File Stats Report

Column Description

FILE Name of the file. The full path name is given if possible.

volume	Name of the logical volume/file system containing the file.
i-node	I-node number for the file within its file system.
opens	Number of times the file was opened while monitored.
total bytes xfrd	Total number of bytes read/written to/from the file.
reads	Number of read calls against the file.
read sizes (bytes)	The read transfer-size statistics (avg/min/max/sdev) in bytes.
read times (msec)	The read response-time statistics (avg/min/max/sdev) in milliseconds.
writes	Number of write calls against the file.
write sizes (bytes)	The write transfer-size statistics.
write times (msec)	The write response-time statistics.
seeks	Number of lseek subroutine calls.

Detailed VM Segment Stats Report

Column	Description
SEGMENT	Internal AIX segment ID.
segtype	Type of segment contents.
segment flags	Various segment attributes.
volume	For persistent segments, the name of the logical volume containing the corresponding file.
i-node	For persistent segments, the i-node number for the corresponding file.
reads	Number of 4096-byte pages read into the segment (that is, paged in).
read times (msec)	The read response-time statistics (avg/min/max/sdev) in milliseconds.

read sequences
Number of read sequences. A sequence is a string of pages that are read (paged in) consecutively. The number of read sequences is an indicator of the amount of sequential access.

read seq. lengths
Statistics describing the lengths of the read sequences in pages.

writes
Number of pages written from the segment (that is, paged out).

write times (msec)
Write response time statistics.

write sequences
Number of write sequences. A sequence is a string of pages that are written (paged out) consecutively.

write seq. lengths
Statistics describing the lengths of the write sequences in pages.

Detailed Logical/Physical Volume Stats Reports

Column
Description

VOLUME
Name of the volume.

description
Description of the volume. (Describes contents if discussing a logical volume; describes type if dealing with a physical volume.)

reads
Number of read requests made against the volume.

read sizes (blks)
The read transfer-size statistics (avg/min/max/sdev) in units of 512-byte blocks.

read times (msec)
The read response-time statistics (avg/min/max/sdev) in milliseconds.

read sequences
Number of read sequences. A sequence is a string of 512-byte blocks that are read consecutively and indicate the amount of sequential access.

read seq. lengths
Statistics describing the lengths of the read sequences in blocks.

writes
Number of write requests made against the volume.

write sizes (blks)
The write transfer-size statistics.

write times (msec)
The write-response time statistics.

write sequences
Number of write sequences. A sequence is a string of 512-byte blocks that are written consecutively.

write seq. lengths
Statistics describing the lengths of the write sequences in blocks.

seeks
Number of seeks that preceded a read or write request, also expressed as a percentage of the total reads and writes that required seeks.

seek dist (blks)
Seek distance statistics in units of 512-byte blocks. In addition to the usual statistics (avg/min/max/sdev), the distance of the initial seek operation (assuming block 0 was the starting position) is reported separately. This seek distance is sometimes very large; so, it is reported separately to avoid skewing the other statistics.

seek dist (cyls)
(Hard files only) Seek distance statistics in units of disk cylinders.

time to next req
Statistics (avg/min/max/sdev) describing the length of time, in milliseconds, between consecutive read or write requests to the volume. This column indicates the rate at which the volume is being accessed.

throughput
Total volume throughput in Kilobytes per second.

utilization
Fraction of time the volume was busy. The entries in this report are sorted by this field in decreasing order.

Flags

-i File
Reads the I/O trace data from the specified File instead of from the real-time trace process. The filemon report summarizes the I/O activity for the system and period represented by the trace file.

Note

Trace data files are usually written in a circular manner. If the trace data has wrapped around, the chronological beginning and end of the trace may occur in the middle of the file. Use the raw mode of the `trcrpt` command to rewrite the data sequentially before invoking the `filemon` command as follows:

```
trcrpt -r file > new.file
```

For the report to be accurate, the trace file must contain all the hooks required by the `filemon` command.

-o File — Writes the I/O activity report to the specified File instead of to the stdout file.

-d — Starts the `filemon` command, but defers tracing until the `trcon` command has been executed by the user. By default, tracing is started immediately.

-T n — Sets the kernel's trace buffer size to n bytes. The default size is 32,000 bytes. The buffer size can be increased to accommodate larger bursts of events, if any. (A typical event record size is 30 bytes.)

Note

The trace driver in the kernel uses double buffering; so, in fact, there will be two buffers allocated of size n bytes. Also, note that these buffers are pinned in memory so that they are not subject to paging. Large buffers may affect the performance of paging and other I/O.

-P — Pins monitor process in memory. The `-P` flag causes the `filemon` command's text and data pages to be pinned in memory for the duration of the monitoring period. This flag can be used to ensure that the real-time `filemon` process is not paged out when running in a memory-constrained environment.

-v — Prints extra information in the report. The most significant effect of the `-v` flag is that all logical files and all segments that were accessed are included in the I/O activity report instead of only the 20 most active files and segments.

-O levels

Monitors only the specified file system levels. Valid level identifiers are:

lf Logical file level

vm Virtual memory level

lv Logical volume level

pv Physical volume level

all Short for lf,vm,lv,pv

Note

The vm, lv, and pv levels are implied by default.

-u

Reports on files that were opened prior to the start of the trace daemon. The process ID (PID) and the file descriptor (FD) are substituted for the file name.

Note

Since PIDs and FDs are reusable, it is possible to see different files reported with the same name field.

Examples

1) To monitor the physical I/O activity of the virtual memory, logical volume, and physical volume levels of the file system, enter:

```
brenzef[root] # filemon
```

The filemon command automatically starts the system trace and puts itself in the background. After this command, enter the application programs and system commands to be run at this time, then enter:

```
brenzef[root] # trcstop
```

After the trcstop command is issued, the I/O activity report is displayed on standard output (but will probably scroll off the screen). The virtual memory I/O report will be limited to the 20 segments that incurred the most I/O.

2) To monitor the activity at all file system levels and write the report to the filemon.out file, enter:

```
brenzef[root] # filemon -o filemon.out -O all
```

The `filemon` command automatically starts the system trace and puts itself in the background. After this command, enter the application programs and system commands to be run at this time, then enter:

```
brenzef[root] # trcstop
```

After the `trcstop` command is issued, the I/O activity report is written to the filemon.out file. All four levels of the file and I/O system (the logical file, virtual memory, logical volume, and physical volume levels) will be monitored. The logical file and virtual memory I/O reports will be limited to the 20 files and segments (respectively) that incurred the most I/O.

3) To monitor the activity at all file system levels and write a verbose report to the filemon.out file, enter:

```
brenzef[root] # filemon -v -o filemon.out -O all
```

The `filemon` command automatically starts the system trace and puts itself in the background. After this command, enter the application programs and system commands to be run at this time, then enter:

```
brenzef[root] # trcstop
```

This example is similar to the previous example, except a verbose report is generated on the filemon.out file. The primary difference is that the `filemon` command will indicate the steps it is taking to start up the trace, and the summary and detailed reports will include all files and segments that incurred any I/O (there may be many) instead of just the top 20.

4) To report on I/O activity captured by a previously recorded trace session, enter:

```
brenzef[root] filemon -i trcfile | more
```

In this example, the `filemon` command reads file system trace events from the input file trcfile. The input file must already be in raw trace format as a result of running the `trcrpt -r` command. Since the trace data is already captured on a file, the `filemon` command does not put itself in the background to allow application programs to be run. After the entire file is read, an I/O activity report for the virtual memory, logical volume, and physical volume levels will be displayed on standard output (which, in this example, is piped to `more`).

5) To monitor the I/O activity for logical and physical volumes only while controlling the monitored intervals using the trcon and trcoff commands, enter:

```
brenzef[root] # filemon -d -o fmon.out -O pv,lv
```

The filemon command automatically starts the system trace and puts itself in the background. After this command, you can enter the unmonitored application programs and system commands to be run at this time, then enter:

```
brenzef[root] # trcon
```

After this command, you can enter the monitored application programs and system commands to be run at this time, then enter:

```
brenzef[root] # trcoff
```

After this command, you can enter the unmonitored application programs and system commands to be run at this time, then enter:

```
brenzef[root] # trcon
```

After this command, you can enter the monitored application programs and system commands to be run at this time, then enter:

```
brenzef[root] # trcstop
```

In this example, the -O flag is used to restrict monitoring to logical and physical volumes only. Only those trace events that are relevant to logical and physical volumes are enabled. Also, as a result of using the -d flag, monitoring is initially deferred until the trcon command is issued. System tracing can be intermittently disabled and reenabled using the trcoff and trcon commands so that only specific intervals are monitored.

Appendix F. Special notices

This publication is intended to help customers and systems engineers to better understand the AIX logical volume manager in order to make better proposals or to develop their storage strategy on RS/6000 systems. See the PUBLICATIONS section of the IBM Programming Announcement for AIX Version 4.3.3 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

This document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX IBM ®

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix G. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

G.1 IBM Redbooks publications

For information on ordering these publications see “How to get IBM Redbooks” on page 397.

- *Understanding IBM RS/6000 Performance and Sizing*, SG24-4810
- *An HACMP Cookbook*, SG24-4553
- *RS/6000 SP System Management Guide*, SG24-5628
- *GPFS: A Parallel file system*, SG24-5165
- *Getting Started with ADSM: A Practical Implementation Guide*, SG24-5416
- *Database Performance on AIX in DB2 UDB and Oracle Environments*, SG24-5511

G.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	**Collection Kit Number**
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

G.3 Other resources

These publications are also relevant as further information sources:

- *AIX Version 4.3 Technical Reference: Base Operating System and Extensions Volume 2,* SC23-4160*.*
- *AIX Versions 4.1 and 4.2 Technical Reference, Volume 2: Base Operating System and Extensions,* SC23-2615
- *AIX Commands Reference, Version 4.3*, SBOF-1877
- *AIX Versions 4.1 and 4.2 Commands Reference*, SBOF-1851
- *AIX Versions 3.2 and 4 Performance and Tuning Guide*, SC23-2365
- *AIX Version 4.3 Technical Reference: Kernel and Subsystems Technical Reference, Volume 1*, SC23-4163.
- *AIX Version 4.3 Problem Solving Guide and Reference*, SC23-4123
- *HACMP Version 4.3 AIX: Installation Guide* , SC23-4278
- *PSSP for AIX Version 3.1 Managing Shared Disks*, SA22-7349
- *HACMP Version 4.3 AIX:Programming Locking Applications*, SC23-4281
- *PCI Adapter Placement Guide*, SA23-2504
- *IBM Recoverable Virtual Shared Disk User’s Guide and Reference*, GC23-3849

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	**Order Number**	**Quantity**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

□ Invoice to customer number

□ Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

AIX. Advanced Interactive eXecutive.

BSD. Berkeley Software Distribution.

C-SPOC. Cluster-Single Point of Control.

CD-ROM. Compact Disc Read Only Memory.

CLVM. Concurrent Logical Volume Manager.

DASD. Direct Access Storage Devices.

FC-AL. Channel arbitrated loop.

GPFS. General Parallel File System.

HACMP. High Availability Cluster Multi-Processing.

HACMP/CRM. High Availability Cluster Multi-Processing, Concurrent Resource Manager.

HACMP/ES. High Availability Cluster Multi-Processing, Enhanced Scalability.

HiPPI. High Performance Parallel Interface Adapter.

I/O. Input/Output.

IBM. International Business Machines.

IPL. Initial Program Load.

ITSO. International Technical Support Organization.

JFS. Journaled File System.

LPSN. Last Physical Sector Number.

LTG. Logical Track Group.

LVDD. Logical Volume Device Driver.

LVID. Logical Volume Identifier.

LVM. Logical Volume Manager.

MWC. Mirror Write Consistency.

MWCC. Mirror Write Consistency Checking.

NBPI. Number of Bytes Per Inode.

ODM. Object Database Manager.

PVID. Physical Volume Identifier.

RAID. Redundant Array of Independent Disks.

RDBMS. Relational Database Management System.

RSCT. Risc System Cluster Technology.

ROS. Read Only Storage.

RVSD. Recoverable Virtual Shared Disk.

SCSI. Small Computer Systems Interface.

SSA. Serial Storage Architecture.

VGDA. Volume Group Descriptor Area.

VGID. Volume Group Identifer.

VGSA. Volume Group Status Area.

VMM. Virtual Memory Manager.

VSD. Virtual Shared Disk.

Index

IBM Redbooks evaluation

AIX Logical Volume Manager, from A to Z: Introduction and Concepts
SG24-5432-00

Your feedback is very important to help us maintain the quality of IBM Redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
_ Customer _ Business Partner _ Solution Developer _ IBM employee
_ None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction __________

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

____

____

____

____

What other Redbooks would you like to see published?

____

____

____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

____

____

____

____

SG24-5432-00
Printed in the U.S.A.

AIX Logical Volume Manager, from A to Z: Introduction and Concepts

SG24-5432-00