Chapter 6

Logical Volume Concepts

This chapter explains the concepts of logical volumes. The use of logical volumes allows one filesystem to spread across multiple disk partitions.

Two types of logical volumes, *lv* and XLV, are supported by IRIX, included in standard system software, and described in this chapter. Support for *lv* logical volumes will be removed from IRIX following IRIX Release 6.2. The procedure for converting from *lv* logical volumes to XLV logical volumes is described in the section "Converting lv Logical Volumes to XLV Logical Volumes" in Chapter 7.

The major sections in this chapter are:

Administration procedures for XLV logical volumes are described in Chapter 7, "Creating and Administering XLV Logical Volumes," and administration procedures for *lv* logical volumes are described in Chapter 8, "Creating and Administering lv Logical Volumes."

Introduction to Logical Volumes

The use of logical volumes enables the creation of filesystems, raw devices, or block devices that span more than one disk partition. Logical volumes behave like regular disk partitions; they appear as block and character devices in the */dev* directory and can be used as arguments anywhere a disk device can be specified.

lv logical volume device files have the names */dev/dsk/lv*<*n*> and */dev/rdsk/lv*<*n*>, where <*n*> is a one or two digit integer. XLV logical volume device files have the names */dev/dsk/xlv/*<*volname*> and */dev/rdsk/xlv/*<*volname*>, where <*volname*> is an alphanumeric string that does not contain periods (dots, ".").

Filesystems can be created, mounted, and used in the normal way on logical volumes, or logical volumes can be used as block or raw devices. Logical volumes provide services such as disk plexing (also known as mirroring) and striping transparently to the applications that access the volumes. Key reasons to create a logical volume are:

- To allow a filesystem or disk device to be larger than the size of a physical disk.
- To increase disk I/O performance.

The drawback to logical volumes is that all disks used in a logical volume must function correctly at all times. If you have a logical volume set up over three disks and one disk goes bad, the information on the other two disks is unavailable and must be restored from backups. However, by using the Disk Plexing Option optional software, you can create multiple copies, called plexes, of the contents of XLV logical volumes, which ensures that all of the information in an XLV logical volume is available even when a disk goes bad.

A logical volume can include partitions from several physical disk drives. By default, data is written to the first disk partition, then to the second disk partition, and so on. Figure 6-1 shows the order in which data is written to partitions in a non-striped logical volume.

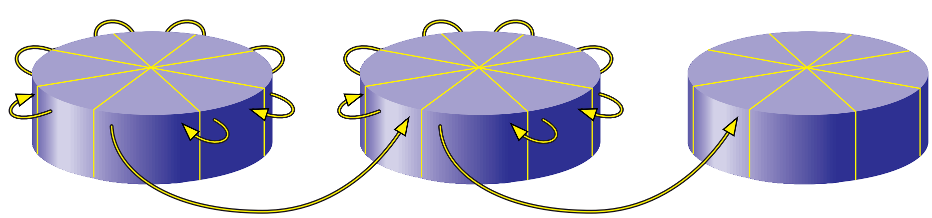

Figure 6-1 Writing Data to a Non-Striped Logical Volume

Also, on striped logical volumes, the volume must have equal-sized partitions on several disks. When logical volumes are striped, an amount of data, called the *stripe unit*, is written to the first disk, the next stripe unit amount of data is written to the second disk, and so on. When each of the disks have been written to, the next stripe unit of data is written to the first disk, the next stripe unit amount of data is written to the second disk, and so on to complete the "stripe." Figure 6-2 shows the order in which data is written to a striped logical volume.

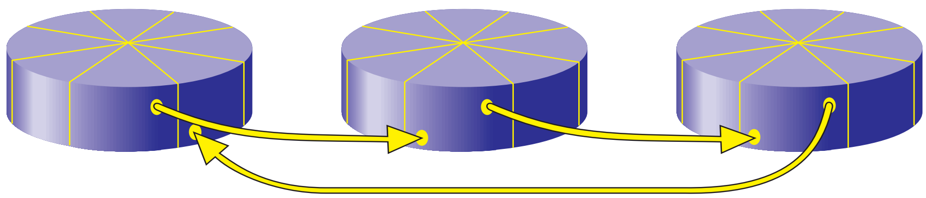

Figure 6-2 Writing Data to a Logical Volume

Because each stripe unit in a stripe can be read and written simultaneously, I/O performance is improved. To obtain the best performance benefits of striping,try to connect the disks you are striping across on different controllers. In this arrangement, there are independent data paths between each disk and the system. However, a small performance improvement can be obtained using SCSI disks striped on the same controller.

There are two basic scenarios for creating logical volumes. In the first scenario, you start with empty disks and perform these basic steps:

1. Create disk partitions as necessary (see "Repartitioning a Disk With fx" in Chapter 2).
2. Create the logical volume.
   - For XLV logical volumes, see the sections "Creating Volume Objects With xlv_make" and "Example 3: A Plexed Logical Volume for an XFS Filesystem With an External Log" in Chapter 7.
   - For *lv* logical volumes, see the sections "Creating Entries in the /etc/lvtab File" and "Creating New Logical Volume With mklv" in Chapter 8.

3. Make a filesystem on the logical volume (see "Making an EFS Filesystem" or "Making an XFS Filesystem" in Chapter 4).

In the second scenario for creating logical volumes, you have a filesystem on a disk partition. You'd like to increase the size of the filesystem ("grow" the filesystem) by creating a logical volume that includes the existing disk partition and a new disk partition. This procedure is explained in the sections "Growing an XFS Filesystem Onto Another Disk" (for XLV logical volumes) and "Growing an EFS Filesystem Onto Another Disk" in Chapter 4 (for *lv* logical volumes).

The next two sections in this chapter describe the features of *lv* and XLV logical volumes.

XLV Logical Volumes

The XLV Volume Manager provides these advantages when XLV logical volumes are used as raw devices and when EFS or XFS filesystems are created on them:

- support for very large logical volumes—up to one terabyte on 32-bit systems and unlimited on 64-bit systems.
- support for disk striping for higher I/O performance
- plexing (mirroring) for higher system and data reliability
- online volume reconfigurations, such as increasing the size of a volume, for less system downtime

However, using XLV logical volumes is not recommended on systems with a single disk.

With XFS filesystems, XLV provides these additional advantages:

- filesystem journal records on a separate partition, which can be on a separate disk, for maximum performance
- access to real-time data

When XFS filesystems are used on XLV volumes, each logical volume can contain up to three subvolumes: data (required), log, and real-time. The data subvolume normally contains user files and filesystem *metadata* (inodes, indirect blocks, directories, and free space blocks). The log subvolume is used for filesystem journal records. It is called an external log. If there is no log subvolume, journal records are placed in the data subvolume (an internal log). Data with special I/O bandwidth requirements, such as video, can be placed on the optional real-time subvolume.

XLV increases system reliability and availability by enabling you to add or remove a copy of the data in the volume (a plex), increase the size of (grow) a volume, and replace failed elements of a plexed volume without taking the volume out of service.

Converting from *lv* logical volumes to XLV logical volumes is easy. Using the commands *lv_to_xlv* and *xlv_make*, you can convert *lv* logical volumes to XLV without having to dump and restore your data.

EFS or XFS filesystems can be made on XLV logical volumes.

Composition of Logical Volumes

Logical volumes are composed of a hierarchy of logical storage objects: volumes are composed of subvolumes, subvolumes are composed of plexes, and plexes are composed of volume elements. Volume elements are composed of disk partitions. This hierarchy of storage units is shown in Figure 6-3, an example of a relatively complex logical volume.

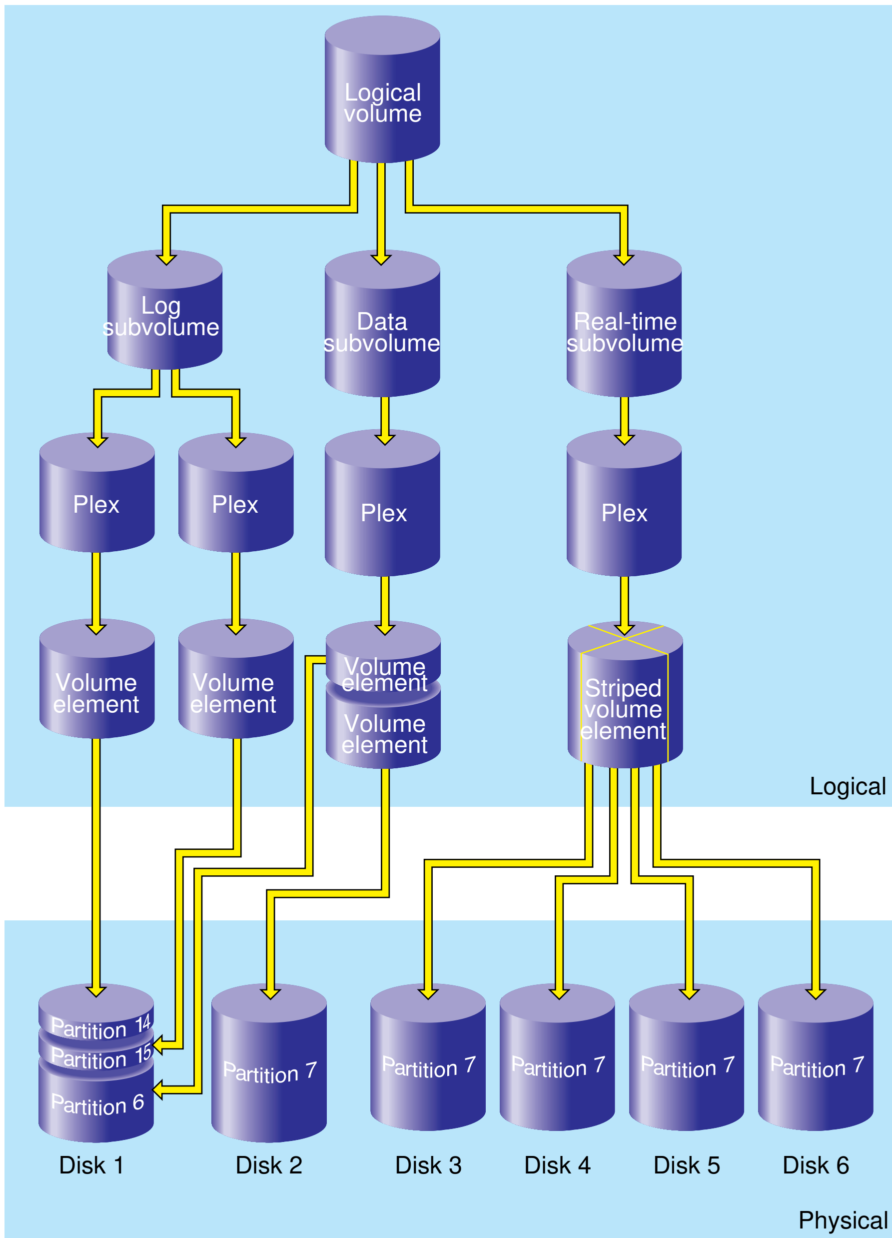

Figure 6-3 Logical Volume Example

Figure 6-3 illustrates the relationships between volumes, subvolumes, plexes, and volume elements. In this example, six physical disk drives contain eight disk partitions. The logical volume has a log subvolume, a data subvolume, and a real-time subvolume. The log subvolume has two plexes (copies of the data) for higher reliability, and the data and real-time subvolumes are not plexed (meaning that they each consist of a single plex). The log plexes each consist of a volume element which is a disk partition on disk 1. The plex of the data subvolume consists of two volume elements, a partition that is the remainder of disk 1 and a partition that is all of disk 2. The plex used for the real-time subvolume is striped for increased performance. The striped volume element is constructed from four disk partitions, each of which is an entire disk.

The subsections below describe these logical storage objects in more detail.

Volumes

Volumes are composed of subvolumes. For EFS filesystems, a volume consists of just one subvolume. For XFS filesystems, a volume consists of a data subvolume, an optional log subvolume, and an optional real-time subvolume. The breakdown of a volume into subvolumes is shown in Figure 6-4.

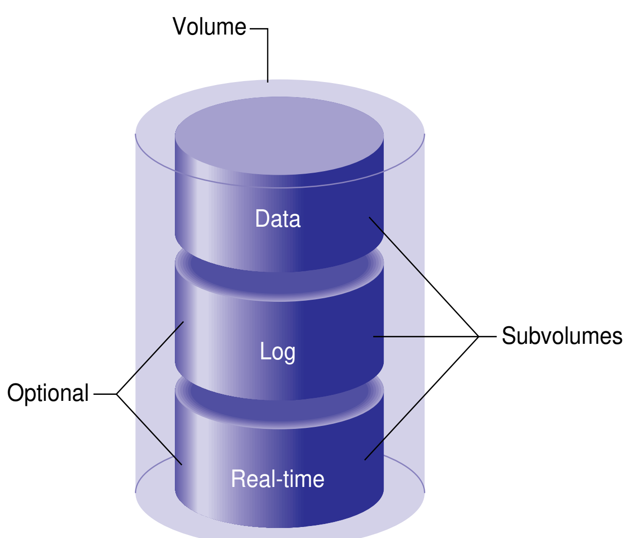

Figure 6-4 Volume Composition

Each volume can be used as a single filesystem or as a raw partition. Volume information used by the system during system startup is stored in logical volume labels in the volume header of each disk used by the volume (see the section "Volume Headers" in Chapter 1). At system startup, volumes won't come up if any of their subvolumes cannot be brought online. You can create volumes, delete them, and move them to another system.

Subvolumes

As explained in the section "Volumes," each logical volume is composed of one to three subvolumes, as shown in Figure 6-5. A subvolume is made up of one to four plexes.

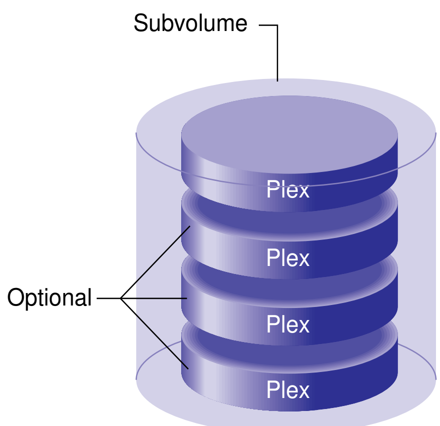

Figure 6-5 Subvolume Composition

Note: The plexing feature of XLV, which enables the use of the optional plexes, is available only when you purchase the Disk Plexing Option software option. See the *plexing* Release Notes for information on purchasing this software option and obtaining the required NetLS license. This NetLS license is installed in a nonstandard location, */etc/nodelock*.

Each subvolume is a distinct address space and a distinct type. The types of subvolumes are:

Data subvolume
: The data subvolume is required in all logical volumes. It is the only subvolume present in EFS filesystems.

Log subvolume — The log subvolume contains XFS journaling information. It is a log of filesystem transactions and is used to expedite system recovery after a crash. Log information is sometimes put in the data subvolume rather than in a log subvolume (see the section “Choosing the Log Type and Size” in Chapter 4 and the mkfs_xfs(1M) reference page and its discussion of the **-l** option for more information).

Real-time subvolume — Real-time subvolumes are generally used for data applications such as video, where guaranteed response time is more important than data integrity. The section “Real-Time Subvolumes” in this chapter and Chapter 9, “System Administration for Guaranteed-Rate I/O,” explain how applications access data on real-time subvolumes.

Subvolumes enforce separation among data types. For example, user data cannot overwrite filesystem log data. Subvolumes also enable filesystem data and user data to be configured to meet goals for performance and reliability. For example, performance can be improved by putting subvolumes on different disk drives.

Each subvolume can be organized independently. For example, the log subvolume can be plexed for fault tolerance and the real-time subvolume can be striped across a large number of disks to give maximum throughput for video playback.

Volume elements that are part of a real-time subvolume should not be on the same disk as volume elements used for data or log subvolumes. This is a recommendation for all files on real-time subvolumes and required for files used for guaranteed-rate I/O with hard guarantees. (See “Hardware Configuration Requirements for GRIO” in Chapter 9 for more information.)

Once a subvolume is created, it cannot be detached from its volume or deleted without deleting its volume. Subvolumes are automatically deleted when their volumes are deleted.

Plexes

A subvolume can contain from one to four *plexes* (also known as *mirrors*). Each plex is an exact replica of all or a portion of the subvolume’s data. By creating a subvolume with multiple plexes, system reliability is increased because there are redundant copies of the data.

If there is just one plex in a subvolume, that plex spans the entire address space of the subvolume. However, when there are multiple plexes, individual plexes can have holes in their address spaces as long as the union of all plexes spans the entire address space. Figure 6-6 shows an example of this. The subvolume contains three plexes. If complete, each plex would be composed of three volume elements. However, two of the plexes are missing a volume element. This is allowed because there is at least one volume element with each address range. In fact, if Plex 1 in the figure were detached (removed from the subvolume), the subvolume would still be functional because there is still at least one volume element with each address range.

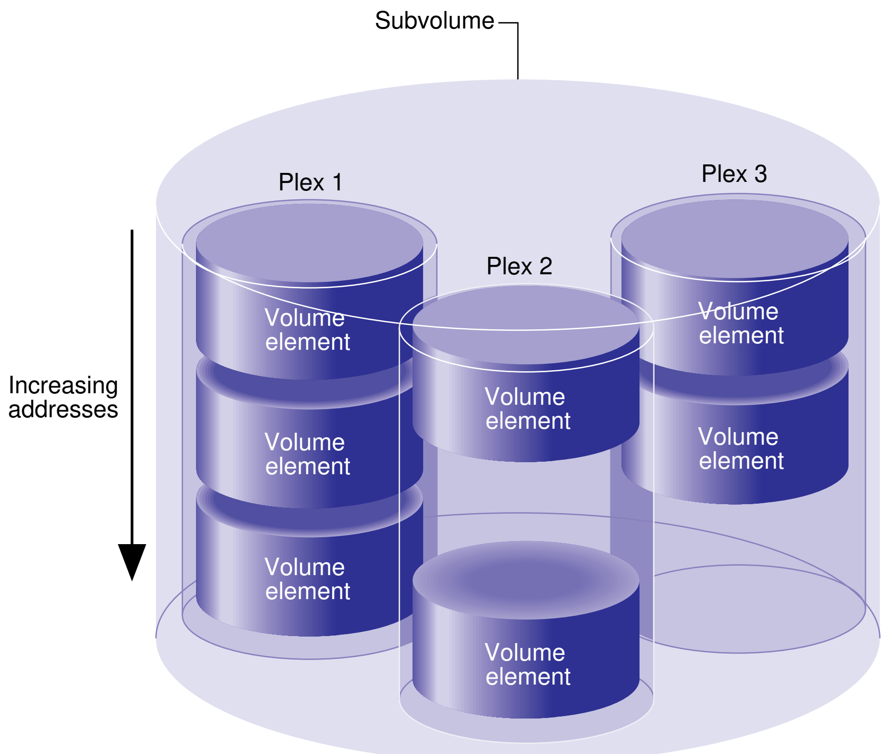

Figure 6-6 Plexed Subvolume Example

Data is written to all plexes. When an additional plex is added to a subvolume, the entire plex is copied (this is called a *plex revive*) automatically by the system. See the xlv_assemble(1M) and xlv_plexd(1M) reference pages for more information.

A plex is composed of one or more volume elements, as shown in Figure 6-7, up to a maximum of 128 volume elements. Each volume element represents a range of addresses within the subvolume.

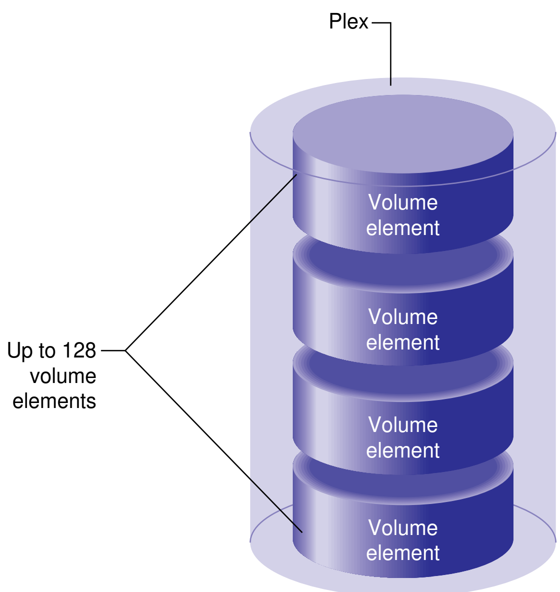

Figure 6-7 Plex Composition

When a plex is composed of two or more volume elements, it is said to have *concatenated* volume elements. With concatenation, data written sequentially to the plex is also written sequentially to the volume elements; the first volume element is filled, then the second, and so on. Concatenation is useful for creating a filesystem that is larger than the size of a single disk.

You can add plexes to subvolumes, detach them from subvolumes that have multiple plexes (and possibly attach them elsewhere), and delete them from subvolumes that have multiple plexes.

Note: To have multiple plexes, you must purchase the Disk Plexing Option software option and obtain and install a NetLS license. See the *plexing* Release Notes for information on purchasing this software option and obtaining the required NetLS license. This NetLS license is installed in a nonstandard location, */etc/nodelock*.

Volume Elements

Volume elements are the lowest level in the hierarchy of logical storage objects: volumes are composed of subvolumes, subvolumes are composed of plexes, and plexes are composed of volume elements. Volume elements are composed of physical storage elements—disk partitions. They provide a way to link one or more disk partitions with or without striping (at least two disk partitions are required for striping).

The simplest type of volume element is a single disk partition. The two other types of volume elements, striped volume elements and multipartition volume elements, are composed of several disk partitions. Figure 6-8 shows a single partition volume element.

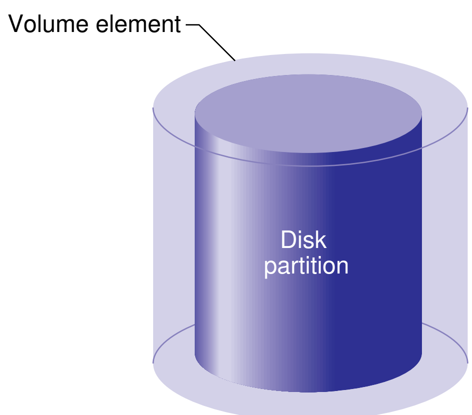

Figure 6-8 Single-Partition Volume Element Composition

Figure 6-9 shows a striped volume element. Striped volume elements consist of two or more disk partitions, organized so that an amount of data called the stripe unit is written to each disk partition before writing the next stripe unit-worth of data to the next partition.

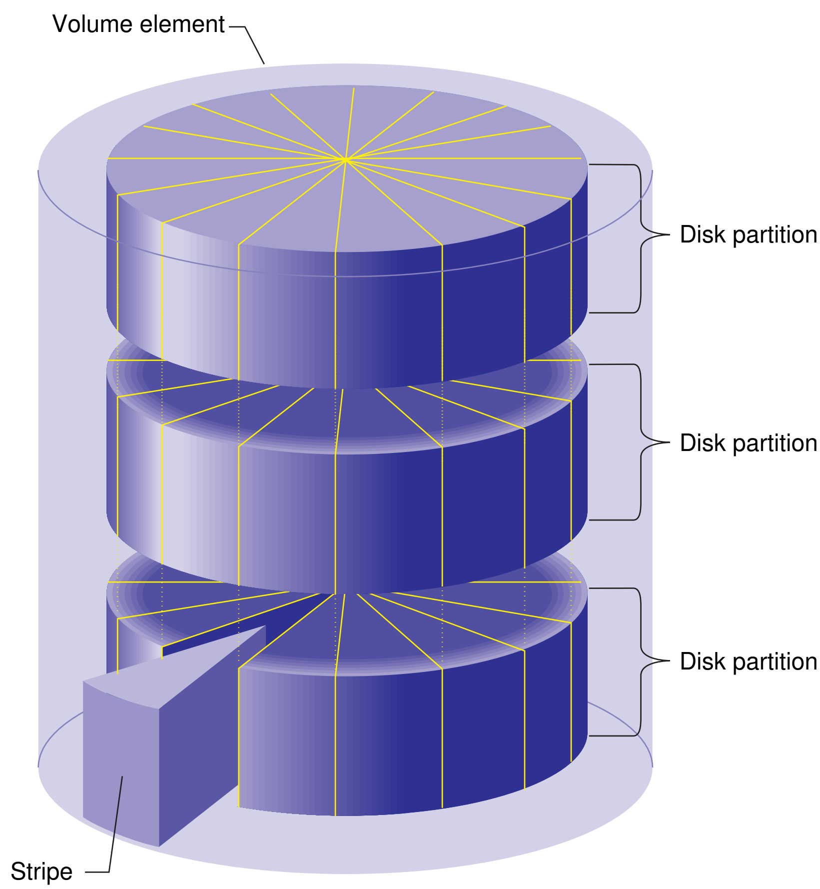

Figure 6-9 Striped Volume Element Composition

Striping can be used to alternate sections of data among multiple disks. This provides a performance advantage by allowing parallel I/O activity. As a rule of thumb, the stripe unit size is a function of the I/O size of the application that uses the striped volume and the number of partitions in the stripe. The stripe unit size should be the application I/O size divided by the number of partitions. The default stripe unit is the device track size, which is generally a good value to use. Stripe unit sizes of less than 32K bytes aren't recommended.

Figure 6-10 shows a multipartition volume element in which the volume element is composed of more than one disk partition. In this configuration, the disk partitions are addressed sequentially.

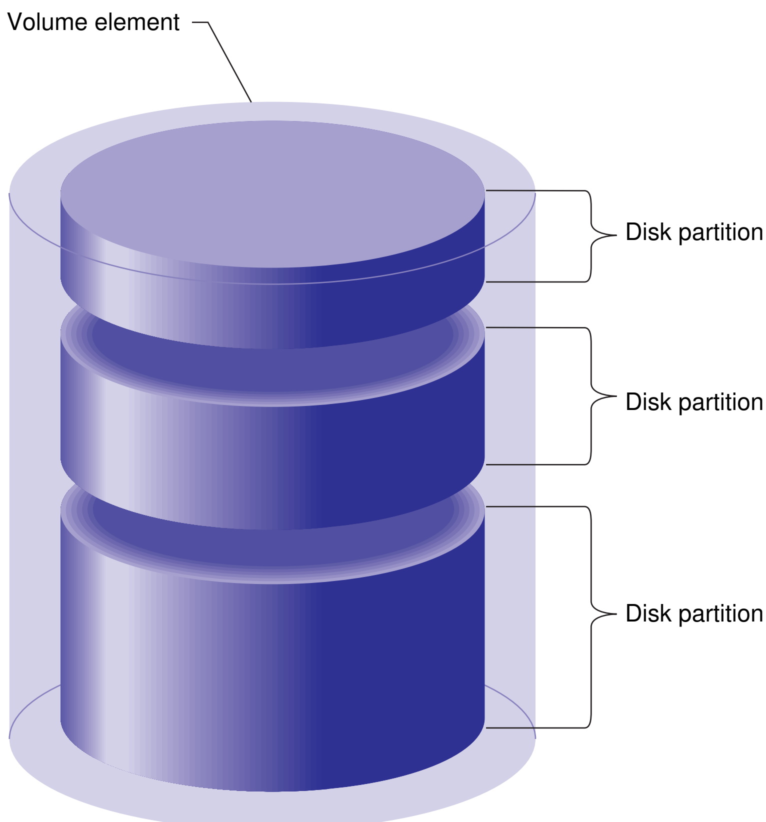

Figure 6-10 Multipartition Volume Element Composition

Any mixture of the three types of volume elements (single partition, striped, and multipartition) can be included in a plex.

XLV Logical Volume Names

Volumes appear as block and character devices in the */dev* directory. The device names for logical volumes are */dev/dsk/xlv/*<*volume_name*> and */dev/rdsk/xlv/*<*volume_name*>, where <*volume_name*> is a volume name specified when the volume is created using the *xlv_make* command.

When a volume is created on one system and moved (by moving the disks) to another system, the new volume name is the same as the original volume name with the hostname of the original system prepended. For example, if a volume called xlv0 is moved from a system called engrlab1 to a system called engrlab2, the device name of the volume on the new system is */dev/dsk/xlv/engrlab1.xlv0* (the old system name engrlab1 has been prepended to the volume name xlv0).

XLV Daemons

The XLV daemons are:

xlv_labd — *xlv_labd* updates logical volume labels. It is started automatically at system startup if it is installed and there are active XLV logical volumes.

xlvd — *xlvd* handles I/O to plexes and performs plex error recovery. It is created automatically during system startup if plexing software is installed and there are active XLV logical volumes.

xlv_plexd — *xlv_plexd* is responsible for making all plexes within a subvolume have the same data. It is started automatically at system startup if there are active XLV logical volumes.

XLV does not require an explicit configuration file, nor is it turned on and off with the *chkconfig* command. XLV is able to assemble logical volumes based solely upon information written in the logical volume labels. During initialization, the system performs a hardware inventory, reads all the logical volume labels, and automatically assembles the available disks into previously defined volumes.

If some disks are missing, XLV checks to see if there are enough volume elements among the available plexes to map the entire address space. If the whole address space is available, XLV brings the volume online even if some of the plexes are incomplete.

XLV Error Policy

For read failures on log and data subvolumes, XLV rereads from a different plex (when available) and attempts to fix the failed plex by rewriting the results. XLV does not retry on failures for real-time data.

For write errors on log and data subvolumes, XLV assumes that these write errors are hard errors (the disk driver and controllers handle soft errors). If the volume element with a hard error is plexed, XLV marks the volume element offline and ignores the volume element from then on. If the volume element is not plexed, the volume element remains associated with the volume and an error is returned.

XLV doesn't handle write errors on real-time subvolumes. Incorrect data is returned without error messages on subsequent reads.

XLV Logical Volume Planning

The following subsections discuss topics to consider when planning a logical volume.

Don't Use XLV When ...

There are some situations where logical volumes cannot be used or are not recommended:

- Raw swap devices cannot be logical volumes. (However, swap space can be added as a regular file in a filesystem and that filesystem could be on a logical volume. See the guide *IRIX Admin: System Configuration and Operation* for more information.)
- Logical volumes aren't recommended on systems with a single disk.
- Striped or concatenated volumes cannot be used for the Root filesystem.

Decide Which Subvolumes to Use

The basic guidelines for choosing which subvolumes to use with EFS filesystems are:

- Only data subvolumes can be used.
- The maximum size of an EFS filesystem is 8 GB, so the data subvolume shouldn't be bigger than that or the space is wasted.

The basic guidelines for choosing which subvolumes to use with XFS filesystems are:

- Data subvolumes are required.
- Log subvolumes are optional. If they are not used, log information is put into an *internal log* in the data subvolume (by giving the **-l internal** option to *mkfs*).
- Real-time subvolumes are optional.

When you want a large raw partition with no filesystem on it, only the data subvolume is used.

Choose Subvolume Sizes

The basic guidelines for choosing subvolume sizes are:

- The maximum size of a subvolume is one terabyte on 32-bit systems (IP17, IP20, and IP22). It is unlimited on 64-bit systems (IP19, IP21, and IP26).
- Choosing the size of the log (and therefore the size of the log subvolume) is discussed in the section "Choosing the Log Type and Size" in Chapter 4. Note that if you do not intend to repartition a disk to create an optimal-size log partition, your choice of an available disk partition may determine the size of the log.

To Plex or Not to Plex?

The basic guidelines for plexing are:

- Use plexing when high reliability and high availability of data are required.
- The Root filesystem can be plexed; each plex must be a single partition volume element.
- Dual-hosted logical volumes (logical volume on disks that are connected to two systems) cannot be plexed.
- RAID disks should not be plexed.
- Plexes can have "holes" in them, portions of the address range not contained by a volume element, as long as at least one of the plexes in the subvolume has a volume element with the address range of the hole.
- The volume elements in each plex of a subvolume must be identical in size with their counterparts in other plexes (volume elements with the same address range). The structure within a volume element (single partition, striped, or multipartition) does not have to match the structure within its counterparts.

- To make volume elements identical in size, use the *fx* command in expert mode (*fx -x*). At the first *fx* menu, give the command **repartition/expert -b**. This enables you to repartition in units of blocks, which will ensure that the volume element is the exact size you want it.

To Stripe or Not to Stripe?

The basic guidelines for striping are:

- The Root filesystem cannot be striped.
- Applications using a striped filesystem should be using direct I/O (see the open(2) reference page).
- Striped disks lead to performance improvement only when the applications that use them make large data transfers that access all of disks in the stripe in the filesystem.
- Striped volume elements should be made of disk partitions that are exactly the same size. When the disk partitions are different sizes, the smallest size is used. Additional space in the larger partitions is wasted.
- For best performance, each disk involved in a striped volume element should be on a separate controller. For some disk types, performance improvement is seen with up to four disks per controller. For other disk types, no additional performance improvement is seen with three or more disk.
- A log subvolume can be striped only if it is an external log. Striping a log does not result in a performance improvement.

Concatenate Disk Partitions or Not?

The basic guidelines for the concatenation of disk partitions are:

- The Root filesystem cannot have concatenated disk partitions.
- It is better to concatenate single-partition volume elements into a plex rather than create a single multipartition volume element. This is not for performance reasons, but for reliability. When one disk partition goes bad in a multipartition volume element, the whole volume element is taken offline.

Real-Time Subvolumes

Files created on the real-time subvolume of an XLV logical volume are known as real-time files. The next three sections describe the special characteristics of these files.

Files on the Real-Time Subvolume and Commands

Real-time files have some special characteristics that cause standard IRIX commands to operate in ways that you might not expect. In particular:

- You cannot create real-time files using any standard commands. Only specially written programs can create real-time files. The next section, "File Creation on the Real-Time Subvolume," explains how.
- Real-time files are displayed by *ls*, just as any other file. However, there is no way to tell from the *ls* output whether a particular file is on a data subvolume or is a real-time file on a real-time subvolume. Only a specially written program can determine the type of a file. The F_FSGETXATTR **fcntl()** system call can determine if a file is a real-time or a standard data file. If the file is a real-time file, the fsx_xflags field of the fsxattr structure has the XFS_XFLAG_REALTIME bit set.
- The *df* command displays the disk space in the data subvolume by default. When the **-r** option is given, the real-time subvolume's disk space and usage is added. *df* can report that there is free disk space in the filesystem when the real-time subvolume is full, and *df –r* can report that there is free disk space when the data subvolume is full.

File Creation on the Real-Time Subvolume

To create a real-time file, use the F_FSSETXATTR **fcntl()** system call with the XFS_XFLAG_REALTIME bit set in the fsx_xflags field of the fsxattr structure. This must be done after the file has first been created/opened for writing, but before any data has been written to the file. Once data has been written to a file, the file cannot be changed from a standard data file to a real-time file, nor can files created as real-time files be changed to standard data files.

Real-time files can only be read or written using direct I/O. Therefore, **read()** and **write()** system call operations to a real-time file must meet the requirements specified by the F_DIOINFO **fcntl()** system call. See the open(2) reference page for a discussion of the O_DIRECT option to the **open()** system call.

Guaranteed-Rate I/O and the Real-Time Subvolume

The real-time subvolume is used by applications for files that require fixed I/O rates. This feature, called guaranteed-rate I/O, is described in Chapter 9, “System Administration for Guaranteed-Rate I/O.”

lv Logical Volumes

lv logical volumes are created and administered by means of a file defining the volumes, */etc/lvtab*, and the commands *mklv*, *lvinit*, *lvinfo*, and *lvck*. There are two components to creating a logical volume from a set of disk partitions:

- Create an entry for the logical volume in the file */etc/lvtab*. */etc/lvtab* is explained in the section “Creating Entries in the /etc/lvtab File” in Chapter 8.
- Run the command *mklv. mklv* writes logical volume information to the volume headers for the disks in the logical volume, creates device files in */dev* for the logical volume, and initializes the logical volume device. Using *mklv* is described in the section “Creating New Logical Volume With mklv” in Chapter 8.

The root partition cannot be part of a logical volume, since the commands required for logical volume initialization must reside on it. Also, swap space cannot be configured as a logical volume.

Striping of *lv* logical volumes imposes some minor restrictions:

- If you want to stripe, all the drives (or to be exact, the partitions used for striping) must be exactly the same size (in disk blocks).
- If you later want to add more disk partitions to the volume, you must add them in units of the striping. That is, if you want to add disks to a three-way striped volume, you must add them three at a time.

Once a logical volume is created, it can be used as if it were a single disk partition. For example, you can create a filesystem on the logical volume and mount the filesystem. The command *lvinfo* prints information about active logical volumes. See the lvinfo(1M) reference page for more information.

The *lvck* command checks the consistency of logical volumes by examining the logical volume labels of devices constituting the volumes. It looks for:

- disks connected in the wrong place
- inconsistencies between the logical volume labels of a logical volume
- internal inconsistencies in */etc/lvtab* entries
- inconsistencies between the logical volume labels of a logical volume and its entry in */etc/lvtab*

The **-d** option of *lvck* can be used to create a new */etc/lvtab* file after disks are moved or renumbered. See the lvck(1M) reference page for details.

lvck has some repair capabilities. If it determines that the only inconsistency in a logical volume is that a minority of devices have missing or corrupt logical volume labels, it is able to restore a consistent logical volume by rewriting good labels. *lvck* queries the user before attempting any repairs on a volume.

Examples of the *lvck* command line are given in the section "Checking Logical Volumes With lvck" in Chapter 8.