



Dynamic Multipathing

**Improving data availability and I/O performance
through intelligent management of storage device access paths**

by

Dhiren Patel
M.G. Venkatesha

with

Christopher Naddeo
Sathish Nayak

VERITAS Software Corporation

April 22, 2005

Scope.....	2
The Importance of Multiple Storage I/O Paths.....	3
Different Forms of Multi-Path Access.....	5
Common Multi-Path Hardware Configurations.....	7
VERITAS Storage Foundation Dynamic Multipathing (DMP)	11
DMP Architecture.....	19
DMP Device Discovery during System Operation.....	22
I/O Path Failover with DMP.....	23
DMP Configuration and Tuning Considerations	25
Using DMP with other Path Managers	29
Conclusion	39

Scope

This paper describes the Dynamic Multipathing (DMP) feature of the VERITAS Volume Manager, a component of the VERITAS Storage Foundation. The DMP architecture described herein was introduced with Volume Manager Version 3.5, and is the current architecture at the time of publication. The paper should be used as a guide to understanding. For up-to-date information on features and coverage, readers are urged to consult VERITAS documentation and support sources.

The Importance of Multiple Storage I/O Paths

The basic techniques for keeping business-critical computer applications and digital data available to users despite hardware and software failures are well-known:

- **Applications.** Applications can be protected against server failures by interconnecting two or more servers to form a cooperative *cluster* controlled by software that enables an application running on any of the servers to *fail over* and restart on another, should its own server fail.
- **Data.** Data can be preserved despite storage device failures by techniques such *mirroring* identical copies on two or more disks,¹ and writing all updates to both simultaneously. Mirroring, sometimes called *RAID-1*, keeps data available if a disk fails, and also improves I/O performance by making two or more disks available satisfy each application read request.

But in enterprise data centers, there is another increasingly important link in the information access chain—the *I/O path* that connects servers with the data they process. The I/O path, represented in Figure 1, is a complex chain consisting of host bus adapter, cables, storage network switch, storage device adapter port, and, in disk arrays, a disk controller.

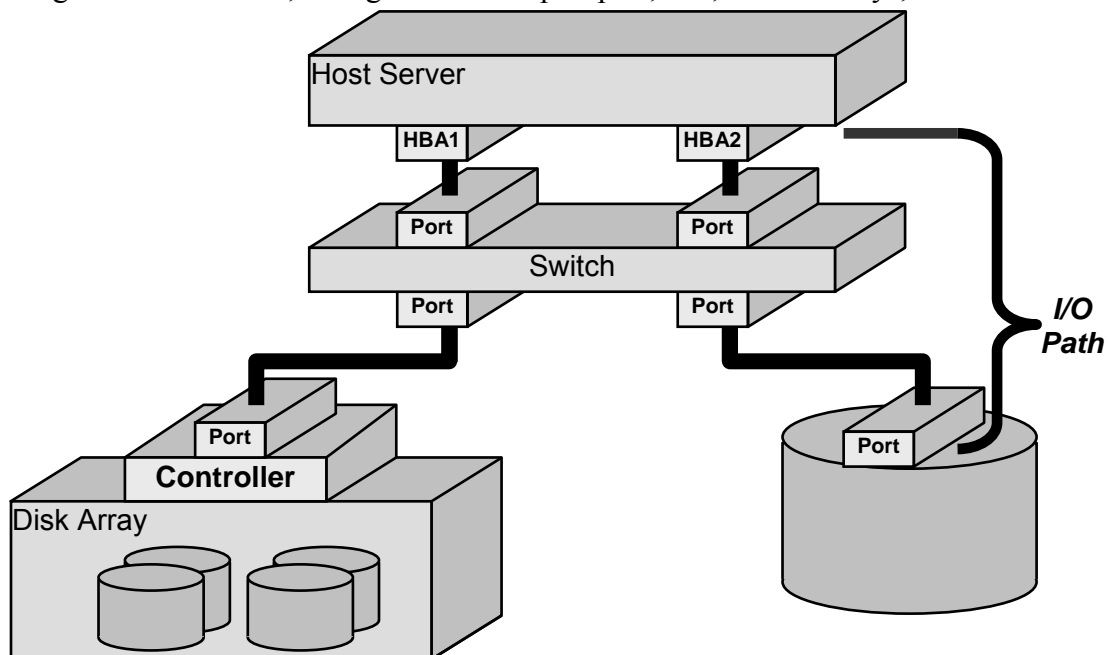


Figure 1: General I/O Path Model

The I/O path shown in Figure 1 begins at a host bus adapter (HBA)² that connects an I/O cable to a server's internal memory access bus. The cable connects the HBA to a corre-

¹ In this paper, the term *disk* refers both to actual disk drives and to the logical units (LUNs) presented to storage network ports by disk arrays.

² Some HBAs have multiple ports, each of which is the starting point of a separate path through the storage network. Since each port is effectively a separate HBA, the model is simplified by treating an HBA as a port.

sponding port in a storage network switch. As Figure 1 suggests, the switch manages logical connections between HBAs and ports within disk array controllers, or between HBAs and disk drives. Disk array controllers, which typically have more than one port, virtualize disks within the array and present them to the storage network as logical units, or LUNs.³

Usage Note

Each unique combination of these elements that can be used to communicate between a host server and a LUN within a disk array or a disk connected directly to the network is a distinct *I/O path*.

Why Multiple I/O Paths?

With increasing deployment of storage networks, IT managers are becoming conscious of the important role that I/O paths play in keeping data available. For example, two disks mirrored by a host-based volume manager may be connected to their hosting server either by the same I/O path, as shown on the left side of Figure 2, or by different paths, as shown on the right. If multiple paths are available, mirroring not only protects against data loss due to disk failure, it also protects against loss of *access* to data if an I/O path element fails, as Figure 2 illustrates.

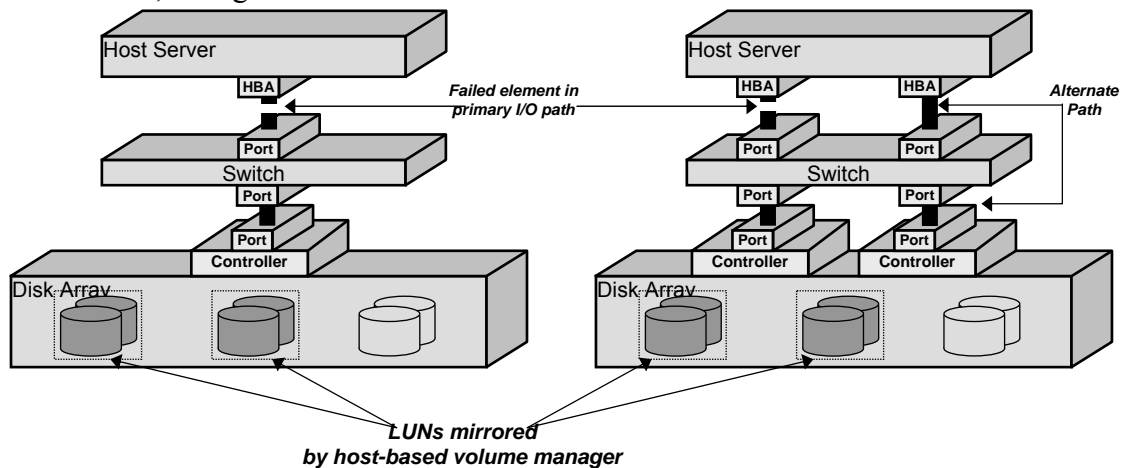


Figure 2: Multiple I/O Paths Improve Data Availability

The server on the left in Figure 2 cannot access its data when the cable between its HBA and the network switch port fails, even though the storage itself remains completely functional, because the cable is a *single point of failure*. The server on the right, on the other hand, can continue to access data if one of its HBAs fails, if a cable fails, or even if one of the disk array's controllers fails, because in each case there is an alternate path that

³ In addition to disk array virtualization, both disks and LUNs are sometimes virtualized by *appliances* or switches within the storage network, and by host-based volume managers such as VxVM. The virtual devices that result from disk array virtualization are universally referred to as LUNs. Virtual devices that result from switch and host-based virtualization are called *virtual disks* or *volumes*.

does not include the failed element.

Thus, a second independent path between server and storage increases the number of component failures an I/O subsystem can withstand without loss of function. But even with an alternate path, I/O path failure can still be tantamount to storage device failure unless the system recognizes that it *has* an alternate path and reroutes I/O requests to it. If a server does not recognize an alternate path to a storage device, the device may as well have failed. Even with failure-tolerant mirrored devices, for example, only devices on still-functioning paths are updated after a path failure. Data redundancy is diminished, even though the unreachable device is still functional. Moreover, I/O performance decreases because one less device is available to satisfy read requests.

Thus, an ability to recognize and utilize alternate I/O paths to storage devices would clearly be preferable. If a path failed, I/O requests would be rerouted to the alternate. Mirrored data would remain fully protected, and the effect on I/O performance would be smaller.

Multiple I/O paths between server and storage device can also improve I/O performance. In many applications, disk arrays satisfy a significant percentage of I/O requests from cache. For example most disk arrays recognize sequential read patterns, and begin to read data into cache in advance of host I/O requests. In this scenario, I/O path bandwidth can actually limit LUN performance. With multiple I/O paths to a LUN however, all can be deliver concurrently as fast as applications request it. Similarly, if an I/O path that provides access to multiple LUNs becomes momentarily overloaded due to activity on one LUN, other LUNs' I/O requests can be rerouted to less-busy paths.

Different Forms of Multi-Path Access

Disks and disk arrays support multi-path access to LUNs in several different ways. Fundamentally, there is a distinction between:

- **Active-active.** If a disk array accepts and executes I/O requests to a single LUN on two or more ports simultaneously, it is called an *active-active (A/A)* array. If a path to an active-active array fails, I/O requests can simply be rerouted to other paths, maintaining continuous access to data stored on the array's LUNs.

EMC's Symmetrix and DMX arrays, Hitachi Data Systems' 9900 Series (Lightning), and IBM's ESS series (Shark) are active-active arrays.

- **Active-passive.** If a disk array accepts and executes I/O requests to a LUN on one or more ports on one array controller (the *primary*), but is able to switch, or "fail over," access to the LUN to alternate ports (*secondaries*) on other array controllers, it is called *active-passive (A/P)*.

EMC's Clariion Cx600 and Cx700, Hitachi Data Systems' 95xx and 9200 series, IBM FAS-T, and Sun's T3 and T4 are active-passive arrays.

In addition to this broad classification, active-passive disk arrays capabilities differ in three ways that affect availability and I/O performance:

- **Multiple primary paths.** If an active-passive array accepts and executes simultaneous

I/O requests to a LUN on two or more ports of the same array controller, it is called an *active-passive concurrent (A/PC)* array. Active-passive concurrent arrays' LUNs fail over to secondary paths on alternate array controllers only when all primary paths have failed. Most active-passive arrays can be configured for active-passive concurrent operation.

EMC's CLARiiON, Hitachi Data Systems' 9500V series, IBM's FASSt-T, and Sun's T3 and T4 are active-passive concurrent arrays.

- **Explicit failover.** Some active-passive arrays fail over from primary I/O paths to secondary ones automatically when they receive an I/O request to a LUN on a secondary path. Others fail over only when they receive special array model-specific SCSI commands from their hosts. Explicit failover simplifies support for active-passive arrays in clusters, where multiple hosts can issue I/O requests directly to LUNs. Without explicit failover capability, cluster software must carefully synchronize all hosts' access to a LUN before initiating implicit failover so that I/O requests from multiple hosts do not result in continuous failovers.⁴

Sun Microsystems T3 and T4 arrays are capable of explicit failover.

- **LUN group failover:** In general, LUNs fail over from one array controller to another individually. Some active-passive arrays, however, can fail administratively defined *groups* of LUNs over together. Arrays with this capability are called *active-passive with group failover capability (A/PG)*. If all primary paths to a LUN in an A/PG array fail, all the LUNs in its group fail over to secondary paths. LUN group failover is faster than failover of individual LUNs, and can therefore reduce the application impact of array controller failure, particularly in disk arrays that present large numbers of LUNs.

Hitachi Data Systems' 9200 series arrays are capable of LUN group failover.

As discussed in later sections, the *dynamic multipathing (DMP)* feature of the VERITAS Volume Manager has a modular architecture that makes it possible to integrate support for new and different types of multi-path access control quickly and easily.

Discovering Multiple I/O Paths

UNIX operating systems “discover” the storage devices that are accessible to them automatically when they start up. Operating system device discovery consists of:

- **Scanning** I/O buses or querying storage network fabrics to determine which bus or network addresses connect to actual disks or LUNs
- **Creating** in-memory data structures in the operating system device tree that identify and describe discovered devices
- **Loading** any specialized drivers required to utilize the devices

⁴ At the time of publication, DMP supports explicit failover of Sun Microsystems' StorEdge T3 and T4 arrays only when they are connected to Solaris servers. Because disk array capabilities and DMP support for them are enhanced frequently, users are advised to consult both VERITAS and disk array vendor product documentation and support information sources for up-to-date information.

At the end of device discovery, an operating system has an in-memory database, or *device tree*, that represents the storage devices with which it can communicate, and has loaded the drivers required to control them.

To an operating system, a storage device is an address on a network that responds appropriately to SCSI storage device commands. UNIX operating systems are not inherently multi-path aware. They view a storage device accessible on two or more paths as two devices at different network addresses. Path management software, such as VERITAS Storage Foundation's Dynamic Multipathing (DMP) discussed starting on page 11, is required to analyze the device tree and identify multi-path devices. DMP's discovery process and the modifications it makes to the operating system device tree are described starting on page 12.

Common Multi-Path Hardware Configurations

The hardware elements that comprise I/O paths can be configured in a variety of ways that affect both system resiliency and I/O performance. The sections that follow describe the most commonly encountered multi-path hardware configurations.

Directly Connected Disk Arrays

Although it is not often encountered in practice, the simplest multi-path hardware configuration consists of a disk or disk array that can present LUNs on two or more ports, each of which is connected directly to a host bus adapter (HBA) on a hosting server. Figure 3 illustrates this configuration.

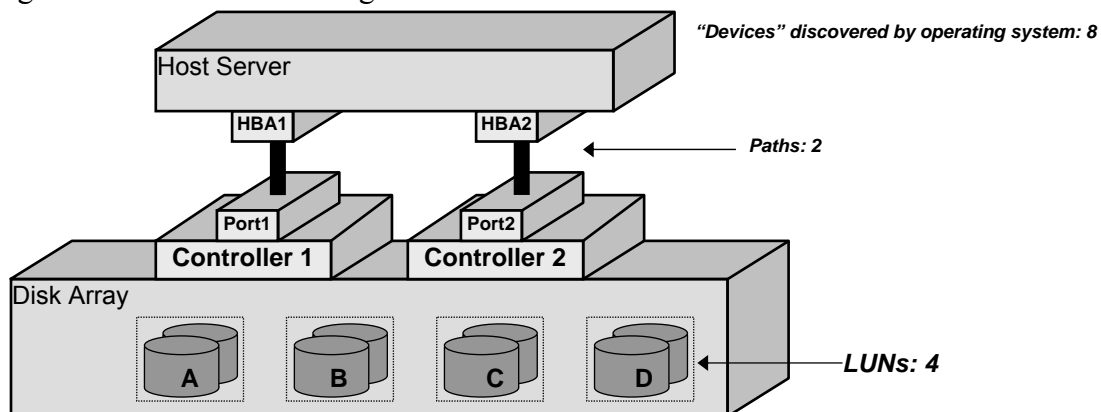


Figure 3: Directly Attached LUNs⁵

The array illustrated in Figure 3 contains four LUNs, each of which is accessible on both of its controller ports. UNIX operating systems would discover the same four LUNs on both paths, so an operating system device tree would contain a total of eight device entries (two for each LUN).

This array might be active-active (able to present LUNs on both ports simultaneously), or active-passive (able to present a LUN on either port, but not on both). If it were active-

⁵ For simplicity, Figure 3 and the figures that follow show artificially small numbers of LUNs.

passive, the array might or might not be capable of explicit failover and LUN group failover. With only one port per controller however, the array could not provide active-passive concurrent LUN access.

Disk Arrays Connected to a Storage Network

A more common multi-path configuration, especially in large data centers, uses a storage network to connect host computers and disk arrays. Figure 4 illustrates this configuration. In the example of Figure 4, each HBA can connect through the switch to each of the disk array's controller ports. There are therefore four unique paths between server and disk array:

- **HBA1**↔**Port1**↔**Port3**↔**Port5**
- **HBA1**↔**Port1**↔**Port4**↔**Port6**
- **HBA2**↔**Port2**↔**Port3**↔**Port5**
- **HBA2**↔**Port2**↔**Port4**↔**Port6**

In this configuration, operating system discovery would report a total of 16 devices (four LUNs on each of the four paths).

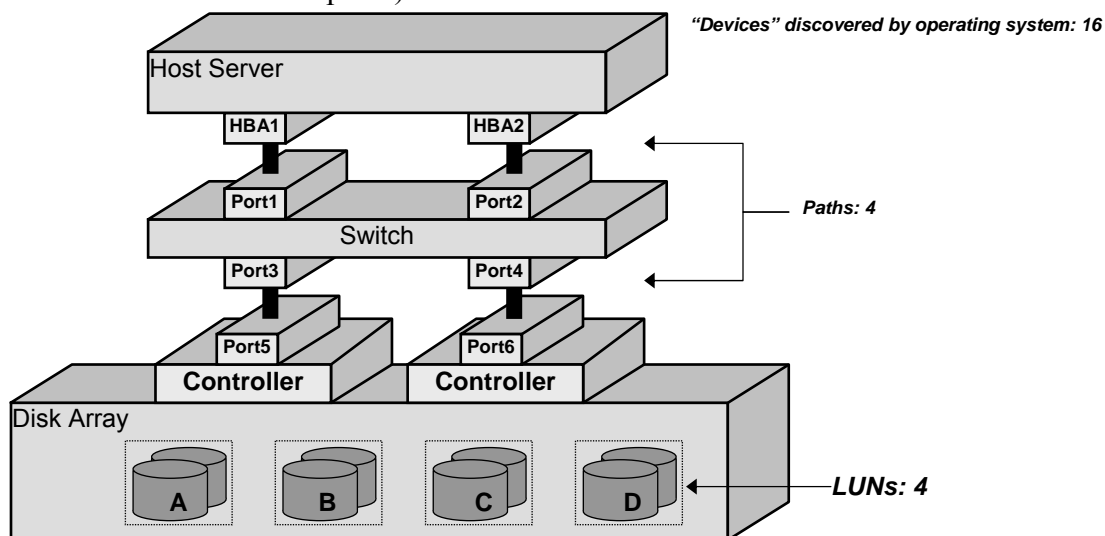


Figure 4: I/O Paths Through a Non-Redundant Storage Network

As with the configuration in Figure 3, the array illustrated in Figure 4 might be active-active or active-passive, with or without explicit and LUN group failover capability. Again, with only one port per controller, active-passive concurrent operation would not be possible. Even if this array were active-passive, concurrent execution of I/O requests to a LUN from both HBAs might be possible, although both would access the same controller port, e.g.:

- **HBA1**↔**Port1**↔**Port3**↔**Port5**
- **HBA2**↔**Port2**↔**Port3**↔**Port5**

This might be slightly advantageous from an availability point of view, since it eliminates failover time for failures of path elements on the host side of the switch, but there is no

performance benefit, because access to any given LUN is limited by the performance of the single controller port on which it is presented.

Disk Arrays Connected to Redundant Storage Networks

A common (and good) storage network design practice, illustrated in Figure 5, is the configuration of identical parallel fabrics connected to the same storage devices and servers, but not to each other. With this configuration, even a complete storage network outage (e.g., a total switch or director failure) leaves all host servers still able to communicate with all storage devices.

In the configuration illustrated in Figure 5, each HBA is connected to a different *fabric* (represented in the figure by a single switch for simplicity). Similarly, each disk array port is connected to a different fabric, creating two paths between any LUN and the host computer:

- HBA1 ↔ Port1 ↔ Port3 ↔ Port5
- HBA2 ↔ Port2 ↔ Port4 ↔ Port6

Operating system discovery would report a total of eight devices (four on each of the two paths).

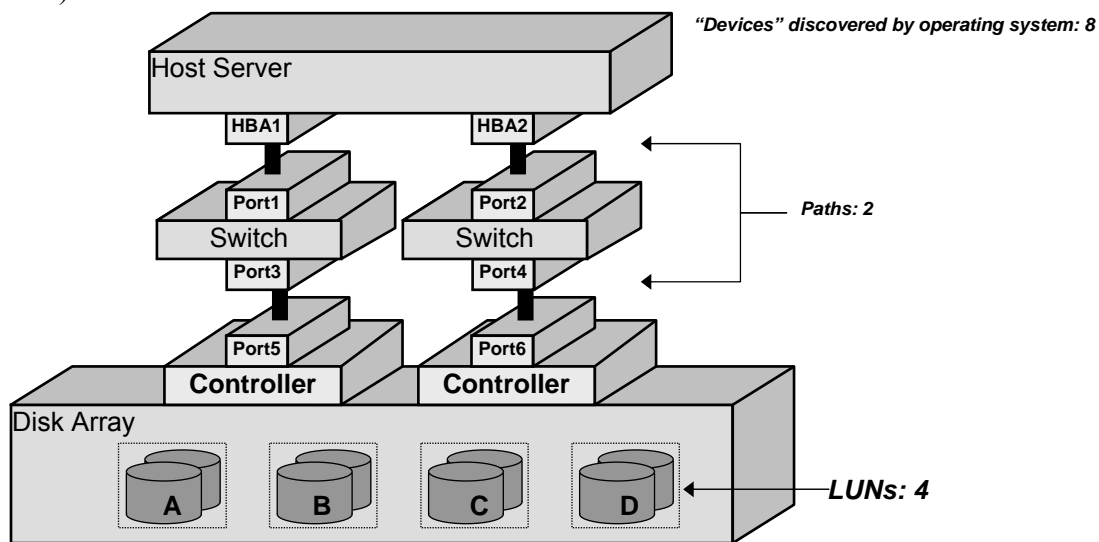


Figure 5: Multiple I/O Paths in a Storage Network with Redundant Fabrics

As in the preceding configurations, this array might be capable of either active-active or active-passive operation. If active-passive, it might be capable of explicit and LUN group failover. But with only one port per controller, active-passive concurrent LUN access would not be possible.

Disk Arrays with Multi-Port Controllers Connected to Multiple Storage Network Fabrics

The configuration shown in Figure 6 is similar to that of Figure 5 in that it includes two parallel fabrics. It differs, however, in that each array controller has two ports. The array can present each LUN on any of four ports (for simplicity, only the port connections for

LUNs B and C are shown). Operating system discovery would report 16 LUNs—each actual LUN would be reported on all four paths:

- HBA1↔Port1↔Port3↔Port5
- HBA1↔Port1↔Port7↔Port9
- HBA2↔Port2↔Port4↔Port6
- HBA2↔Port2↔Port8↔Port0

The disk array in Figure 6 might be active-active or active-passive, and if active-passive, might be capable of explicit failover and LUN group failover. Because each array controller has two ports, active-passive concurrent operation is possible. LUNs might be presented on primary ports 5 and 9, for example, with ports 6 and 0 designated as secondary ports. EMC Clariion Cx700 arrays can be configured in this fashion.

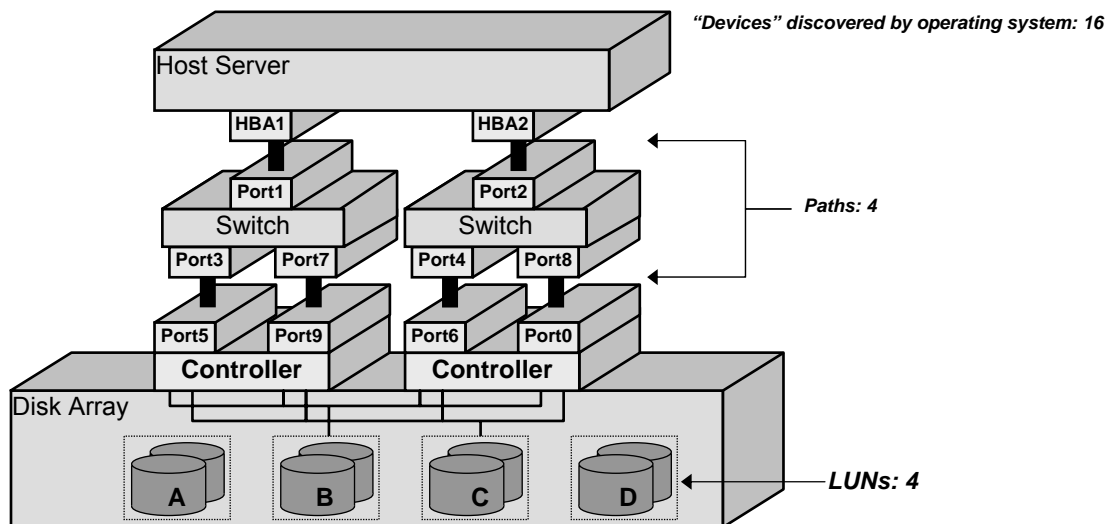


Figure 6: Multi-Port Controllers Connected to Redundant Fabrics

Disk Arrays with Multi-Port Controllers Cross-Connected to Redundant Fabrics

The configuration illustrated in Figure 7 is identical to that of Figure 6 with the exception that each disk array controller is connected to both fabrics. Each LUN can be accessed on any of the four ports (again, for simplicity, Figure 7 illustrates only the LUN B and C port connections.)

Operating system discovery would report each LUN on four paths:

- HBA1↔Port1↔Port3↔Port5
- HBA1↔Port1↔Port7↔Port6
- HBA2↔Port2↔Port4↔Port9
- HBA2↔Port2↔Port8↔Port0

From a path management point of view, this configuration is identical to the preceding one. It might offer slightly better availability since a controller failure would still leave

both fabrics usable (unlike the configuration in Figure 6). Similarly, failure of a fabric would leave the disk array able to use both of its controllers, eliminating the need for path failover.

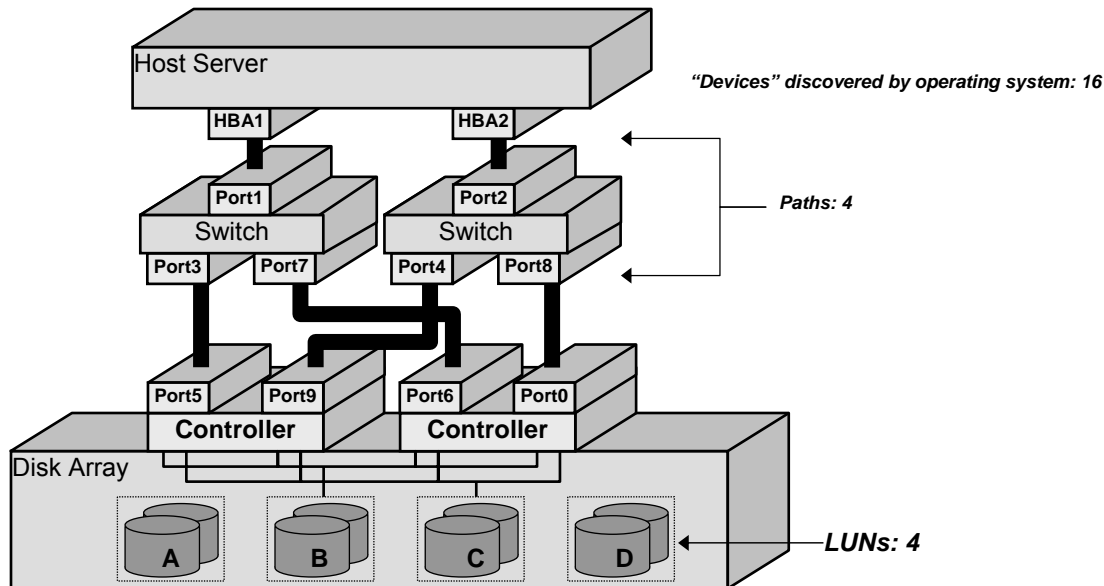


Figure 7: Multi-Port Controllers Cross-Connected to Redundant Fabrics

VERITAS Storage Foundation Dynamic Multipathing (DMP)

Effective use of multiple I/O paths requires both awareness of storage network topology and an ability to execute pre-defined policies automatically in response to rapidly changing conditions in the I/O subsystem. The VERITAS Storage Foundation's *Dynamic Multipathing* (DMP) feature automates the management of multiple I/O paths between servers and storage devices in accordance with pre-defined administrative policies. DMP enhances I/O subsystem availability and I/O performance in three ways:

- **Data availability.** If an I/O path to a multi-path storage device fails, DMP automatically reroutes I/O requests to an alternate path transparently to applications and without administrator intervention. When a failed path returns to service, DMP restores the original path configuration automatically and transparently as well.
- **I/O performance.** For disk arrays that support simultaneous access to a single storage device on multiple paths, DMP enhances I/O performance by distributing I/O requests across all available paths according to pre-defined load balancing policies.
- **Application resiliency.** In cluster configurations, DMP improves application availability by eliminating application failovers that would otherwise result from I/O path failures.

In addition to these, DMP makes it possible to manage both storage virtualization and I/O path policies from a single console or graphical interface because it is part of the

VERITAS Storage Foundation Volume Manager (VxVM).

DMP In the UNIX Storage I/O Software Stack

DMP is a layer in the UNIX storage I/O software stack. While different platforms' implementations differ in detail, UNIX I/O software stacks share a common overall structure, simply because all perform the same basic functions to provide I/O services. Figure 8 shows a simplified model of a generic UNIX storage I/O software stack that includes VxVM and DMP.

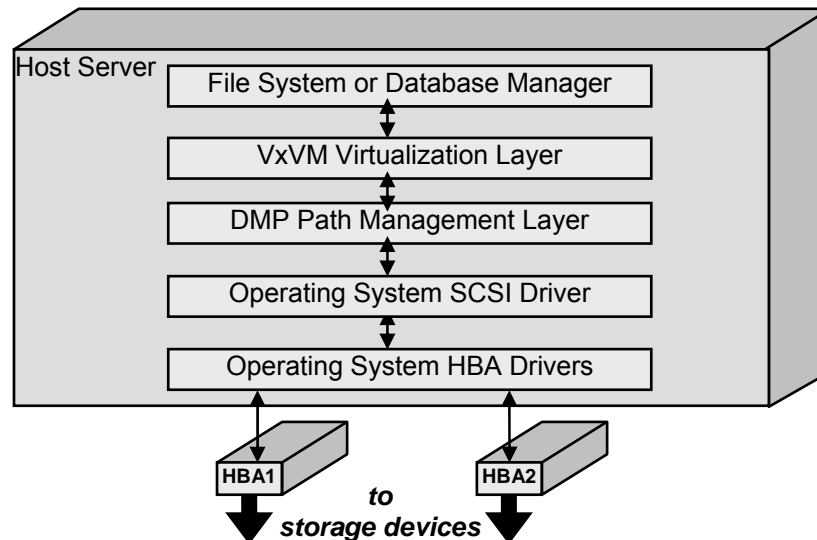


Figure 8: Generic Model of the UNIX Storage I/O Software Stack

In a typical server, almost all I/O requests to a server's I/O subsystem are issued by a file system (in some cases, database managers issue I/O requests to "raw" storage). File systems issue their I/O requests to VxVM virtual volumes (e.g., `/dev/vx/rdisk/diskgroup/volume`). The VxVM virtualization layer converts them into equivalent requests to physical disks or LUNs. For example, if a file system issues a write request to a mirrored volume, the VxVM virtualization layer converts it into write requests to corresponding block ranges of each of the mirrors that comprise the volume. Path management software like DMP necessarily occupies a position below virtualization in the I/O stack. It receives I/O requests from the VxVM virtualization layer, determines which path should carry each one, and issues it to the operating SCSI system driver on that path. UNIX operating systems have two layers of storage I/O drivers—a SCSI layer that converts operating system I/O request structures into SCSI command data blocks (CDBs) and one that sends and receives messages containing CDBs and data on the storage network or I/O bus.

DMP and Device Discovery

As discussed earlier (page 6), UNIX operating systems *discover* the I/O devices that are accessible to them when they start up. Following operating system discovery, the VxVM configuration daemon, `vxconfigd`, discovers information that VxVM requires and cre-

ates in-memory data structures that describe (among other things) devices' multi-path capabilities.

DMP Multi-Path Devices in the Operating System Device Tree

For each disk or LUN it detects, a UNIX operating system creates data structures sometimes called *nodes* or *device handles*, in its device tree. For example, the Solaris operating system creates nodes in both the `/dev/rdisk` and `/dev/dsk` paths for each device it detects. If a device is accessible on two or more paths, operating systems treat each path as a separate device, and create nodes corresponding to each path.

During its discovery process, VxVM's `vxconfigd` daemon creates similar structures called *metanodes* in the `/dev/vx/rdmp` and `/dev/vx/dmp` trees for each storage device it detects. Each metanode represents a *metadevice*, a VxVM abstraction that corresponds to a disk or LUN and all the I/O paths on which it can be accessed. The VxVM virtualization layer issues its I/O requests to these metadevices.

The `vxconfigd` daemon identifies multiple paths to a device by issuing a SCSI inquiry command to each operating system device. A disk or LUN responds to a SCSI inquiry command with information about itself, including vendor and product identifiers and a unique serial number. An administrator can use the command `/etc/vx/diag.d/vxdmpinq` to issue a SCSI inquiry to a device and display the response, as Dialog 1 illustrates.

```

dcsun51 $/etc/vx/diag.d/vxdmpinq /dev/vx/rdmp/HDS9970V0_4s2
Inquiry for /dev/vx/rdmp/HDS9970V0_4s2, evpd 0x0, page code 0x0, flags 0x4
Vendor id           : HITACHI
Product id         : OPEN-9       -SUN
Revision          : 2106
Serial Number      : 045175F30009

```

Dialog 1: Information Returned by SCSI Inquiry Command

If two operating system devices respond to SCSI inquiry commands with the same serial number, they are in fact the same physical disk or LUN responding on two different paths.⁶

If VxVM discovery encounters only one instance of a particular serial number, the device can only be accessed on a single path. DMP links its metanode for each single-path device to the corresponding node in the operating system tree, as Figure 9 illustrates, and marks the device for “fast path” access by the VxVM virtualization layer. During system operation, the VxVM virtualization layer sends I/O requests to fast-path devices directly to the operating system's SCSI driver without passing them to DMP.

⁶ A consequence of this method of detecting multiple paths to a device is that DMP can only support disks and LUNs that return the same unique disk identifier in response to SCSI inquiry commands on all paths. This is generally true for path management software.

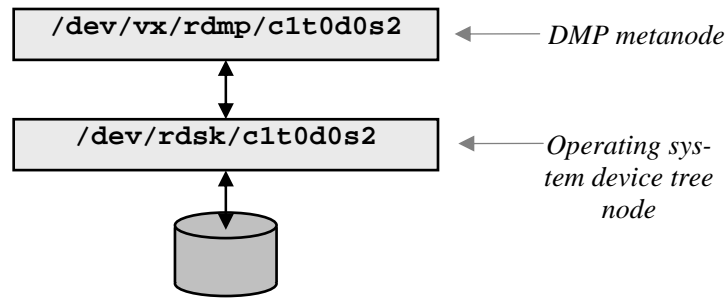


Figure 9: VxVM Subtree for a Single-Path Device (Solaris)

A device that is accessible on multiple paths returns the same serial number to inquiry commands on all paths. When DMP encounters the same serial number on different paths, it creates a metanode and links it to all operating system nodes that represent paths to the device, as Figure 10 illustrates.

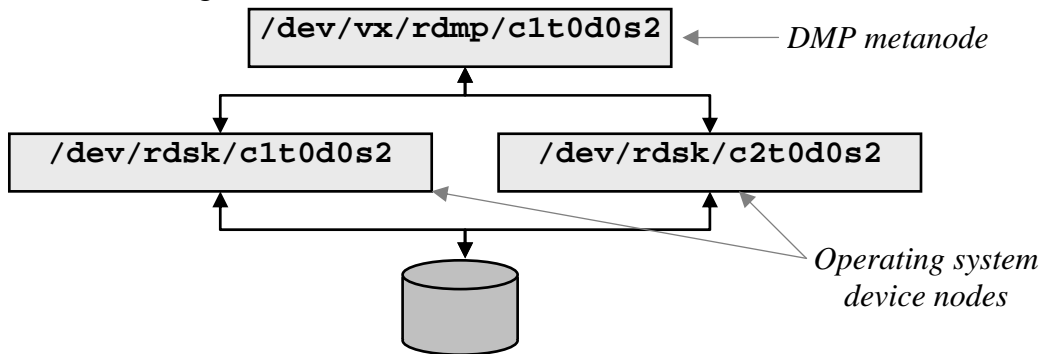


Figure 10: VxVM Subtree for a Dual-Path Device (Solaris)

An administrator can use the `vxdisk path` command to display information about VxVM metadevices and the paths to which they correspond, as Dialog 2 illustrates.

# vxdisk path				
SUBPATH	DANAME	DMNAME	GROUP	STATE
c1t0d0s2	c1t0d0s2	mydg01	mydg	ENABLED
c2t0d0s2	c1t0d0s2	mydg01	mydg	ENABLED
c1t1d0s2	c1t1d0s2	mydg02	mydg	ENABLED
c2t1d0s2	c1t1d0s2	mydg02	mydg	ENABLED

Dialog 2: `vxdisk path` Command for Multi-Path Disks

For the dual-path device illustrated in Figure 10, the `vxdisk path` command in Dialog 2 shows the metanode `c1t0d0s2` (in the `/dev/vx/rdmp` subtree) as corresponding to operating system nodes `c1t0d0s2` and `c2t0d0s2`. Information displayed by the VxVM `vxdisk path` command includes:

- The *disk access name* (**DANAME**, or VxVM metanode name, e.g., `c1t0d0s2` in Dialog 2) of each metadvice. The **DANAME** is the name used by the operating system to manage the LUN.
- The *disk media name* (**DMNAME**, or VxVM user-friendly device name, e.g., `mydg01`

in Dialog 2), of each metadvice. The **DMNAME** is used in VxVM management operations.

- The operating system device nodes (**SUBPATHS**) corresponding to each metadvice (e.g., **c1t0d0s2** and **c2t0d0s2** corresponding to VxVM metanode **c1t0d0s2** in Dialog 2)
- The VxVM disk group membership of metadvice (**mydg** in Dialog 2)
- The operational state of each metadvice on all access paths. Dialog 2 indicates that all paths are **ENABLED**, or eligible to handle I/O requests. Paths may also be **DISABLED**, either by administrative command, or by DMP itself if it fails to recover from an I/O error.

DMP I/O Load Balancing Policies

In most instances, DMP is installed primarily to increase data availability by keeping storage devices accessible when I/O paths fail. When all paths are operational, however, DMP can also improve a device's I/O performance by routing each request to the most appropriate path. Using structures similar to that shown in Figure 10, DMP can choose the optimal path on which to route each file system I/O request.

The "optimal" path to a device can change over time based on I/O load, but path selection can also be a matter of system policy. At the time of publication, DMP includes six different I/O request routing policies that can be applied to multi-path storage devices to influence the balancing of I/O requests across paths. The paragraphs that follow describe these six policies.

Balanced Path Routing

DMP's balanced path policy routes I/O requests to paths based on the starting block addresses they specify. Effectively, this policy divides a device's block address space into as many disjoint regions as there are active paths, and assigns each I/O request to a path that corresponds to the region in which the data it transfers falls.

For LUNs using the balanced path policy, DMP divides the starting data address specified each I/O request by the system-wide parameter **DMP_PATHSWITCH_BLK_SHIFT** and discards the remainder. The quotient of the division modulo the number of active paths is used to index the active path used to issue the I/O command.

As an example, Figure 11 illustrates the balanced path I/O policy for an active-active device with two paths. For graphic simplicity, **DMP_PATHSWITCH_BLK_SHIFT** has an artificially low value of 4. In this example, DMP would route read and write requests that specify a starting block addresses between 00 and 03 to path **c1t0d0s0**, those that specify one of blocks 04-07 to path **c2t0d0s0**, those that specify one of blocks 08-11 to path **c1t0d0s0**, and so forth.

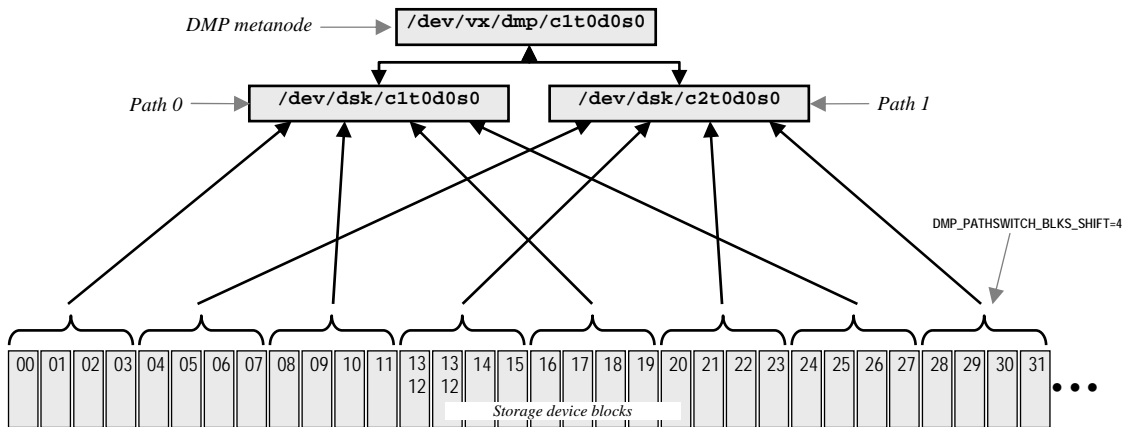


Figure 11: Balanced I/O Policy Path Selection

To illustrate the general algorithm, for a read or write request specifying a starting address of block 13, DMP would divide the address by **DMP_PATHSWITCH_BLK_SHIFT** (13/4), giving an integer quotient of three. Three modulo the number of active paths (two) is one, so DMP would issue an I/O request to the operating system SCSI driver on path 1 (operating system device **c2t0d0s0**).

The balanced path policy is DMP's default policy for active-active arrays' LUNs (in earlier versions of DMP, it was the *only* available policy). It is particularly useful for high-speed sequential reading from active-active disk arrays and dual-port disk drives with read-ahead cache. Aligning the value of **DMP_PATHSWITCH_BLK_SHIFT** with the sequential I/O request size causes DMP to route successive requests to alternate paths, which frequently allows data for two or more requests to transfer concurrently.

The default value for **DMP_PATHSWITCH_BLK_SHIFT** is 2048 blocks, or 1 megabyte. The value can be overridden for individual arrays by using the **setattr** option of the **vxddmpadm** command. Overriding the global **PATHSWITCH_BLK_SHIFT** value is useful in systems connected to two or more different types of arrays.

Round-Robin Routing

The round-robin I/O request routing policy attempts to issue an equal number of I/O requests on each active I/O path to a device. For each request, DMP computes a pseudo-random number and assigns a path based on the computed number modulo the number of active paths.

The round-robin policy is useful when most I/O requests to a LUN specify approximately the same amount of data transfer, and in storage networks whose loading is relatively evenly distributed. Round robin is the default DMP policy for active-passive concurrent arrays with multiple primary paths enabled.

Minimum Queue Length Routing

The minimum queue length policy routes each I/O request to the active path with the

smallest number of outstanding requests. Each time DMP assigns a request to a path, it increments the path's outstanding request counter. Each time a request completes, the path's request counter is decremented. For each new request, DMP selects the path with the smallest outstanding request counter value. This policy tends to counteract momentary load imbalance automatically, as for example, when a path bottlenecks because of error retries or overload from other LUNs.

Adaptive Routing

The adaptive routing policy allows DMP to dynamically route I/O requests based on calculated path priorities. When this policy is in effect, DMP records the service time and amount of data transferred for each request, and periodically calculates a priority for each path based on its recent throughput (bytes per second). The priority calculation algorithm produces higher priorities for paths that have recently delivered higher throughput. Incoming I/O requests are routed to paths in proportion to the paths' relative priorities. For example, if there are three active paths whose priorities are calculated as 3, 2, and 1 respectively, half of incoming requests are routed to path 1, a third to path 2, and the remaining sixth to path 3. As total I/O load on higher priority paths increases, the paths tend to deliver lower throughput, resulting in lower priorities on the next recalculation cycle.

The adaptive policy is useful with rapidly varying I/O loads, such as database applications that include both transactions (short transfers) and periodic table scans (long transfers). It is also useful in storage networks where different paths have discernibly different average performance, such as paths with different numbers of network "hops" or individual links of different speeds.

Priority Routing

With the priority routing policy, DMP routes requests based on path priority as with the adaptive policy. Path priorities are assigned by administrators rather than calculated by DMP, however, and do not change without administrative action. The priority routing policy allows administrators to assign path priorities based on considerations other than performance, such as applications' relative importance to an enterprise.

Single Active Path (Preferred Path) Routing

As its name implies, the single active path policy causes DMP to route all I/O requests to one path (called the *preferred* path). Only if the preferred path fails does DMP route I/O to a secondary one. The single active path policy is the default for non-concurrent active-passive arrays. If this policy is configured for a LUN in an active-active array, DMP routes all I/O requests to the single active path; other paths are not used unless the active one fails.

Usage Note

I/O performance of active-passive arrays can be influenced by the assignment of different LUNs' preferred paths to different controllers. For example, in an array with two controllers, odd numbered LUNs might be assigned to one controller and even numbered LUNs to the other. If certain LUNs are known a priori to be heavily loaded, their primary path assignments can be distributed across controllers.

Determining the Effect of DMP Load Balancing Policies

Administrators can monitor the effect of any of these load balancing policies by using the `vxddmpadm iostat` command, as Dialog 3 illustrates.

```
dcsun51 $vxddmpadm iostat show all
cpu usage = 19733393us      per cpu memory = 32768b
OPERATIONS                MBYTES                AVG TIME(ms)
PATHNAME                 READS                 WRITES                READS                 WRITES                READS                 WRITES
c0t1d0s2                  159                   0                      79                    0    10.670886  0.000000
c0t0d0s2                   20                    7                     1220                   162    0.042623  0.203704
c2t3d13s2                  870                   236                   17426                   2158    0.205153  0.101946
c3t3d13s2                  334                    4                     15684                    56    0.173871  0.017857
c2t3d12s2                 9127                   507                    9236                    18365    0.251299  0.076940
c3t3d12s2                   45                    649                    255                    18632    0.152941  0.074281
c2t3d11s2                 1311                    11                    2068                    185    0.133946  0.021622
c3t3d11s2                   0                      1                      0                       8    0.000000  0.000000
c2t3d10s2                1241887                1200897                19851284                19849637    0.306276  0.213964
c3t3d10s2                1241300                1285848                19850538                19968007    0.274636  0.190663
c2t3d9s2                  1240586                1200829                19849347                19850382    0.288218  0.215717
c3t3d9s2                  1240839                1204491                19852391                19886690    0.255909  0.190155
c2t3d8s2                  1240585                1200814                19849319                19850598    0.272347  0.241037
c3t3d8s2                  1241296                1199258                19850357                19878681    0.241508  0.213279
c2t3d7s2                  1240584                1201024                19849315                19850304    0.293359  0.242801
c3t3d7s2                  1246021                1201910                19846281                19881291    0.262374  0.215197
c2t3d6s2                   1311                    12                    2068                    161    0.132495  0.031056
c3t3d6s2                   0                      1                      0                       8    0.000000  0.000000
c2t3d5s2                   10                     18                     131                    800    0.473282  0.013750
c3t3d5s2                   0                      1                      0                       8    0.000000  0.000000
c2t3d4s2                 17059295                11930299                137101333                76064118    0.102189  0.158222
c3t3d4s2                   6703                  2242672                 625389                 2373697    0.034997  0.899039
c2t3d3s2                   82888                  1923459                 652854                 8340732    0.291318  0.180282
c3t3d3s2                   82119                  913373                  549421                 4490416    0.357065  0.189884
c2t3d2s2                   21                     0                      10                      0     5.600000  0.000000
c3t3d2s2                   0                      0                      0                       0     0.000000  0.000000
c2t3d1s2                   21                     0                      10                      0     6.400000  0.000000
c3t3d1s2                   0                      0                      0                       0     0.000000  0.000000
c2t3d0s2                   21                     0                      10                      0     5.400000  0.000000
c3t3d0s2                   0                      0                      0                       0     0.000000  0.000000
```

Dialog 3: DMP Collection of I/O Statistics

The output of the `vxddmpadm iostat` command displays both the number of read and write operations, the amount of data read and written, and the average execution time for reads and writes for each path since VxVM's `iostat` daemon was started, or since its

counters were last reset (using a variant of the same command). The command can be executed at intervals to determine the efficacy of a given load balancing algorithm under actual system I/O loads.

DMP Architecture

Because its value is greatest in large enterprise data centers, DMP is more often used with disk array LUNs than with directly attached disks. Although disks and disk arrays adhere to standards for data transfer (SCSI, Fibre Channel and iSCSI), each disk array model has its own unique way of controlling multi-path LUN and disk access. To support a particular disk array model, DMP must be customized to handle the array's multi-path access capabilities and to interact properly with its interface protocols. The need to support new disk array models as they reach the market rather than on VxVM release cycles prompted the introduction of a unique array support architecture in Version 3.2 of VxVM. This architecture keeps DMP integrated with VxVM, but at the same time makes it easy to add multi-path access support for new disk array models rapidly and efficiently. The DMP array support architecture is available in the form of a software development kit (SDK) that *VERITAS Enabled* partners can use to develop DMP support for new disk array models independently of VxVM releases.

DMP Support for Different Disk Array Models

Disk arrays with multi-path LUN access capability may be supportable by DMP without custom software. DMP can manage multi-path access to a disk array's LUNs by treating them as disks, provided that the array has the following properties:

- Multi-path access to LUNs is active-active
- LUNs respond to SCSI inquiry commands with unique serial numbers, and each LUN's serial number is reported identically on all paths
- LUNs' unique serial numbers can be read from the SCSI standard mode page location

If an array has these properties, the `vxddladm` command with the `addjbod` option can be used to add its LUNs (identified by the vendor ID and product ID reported in response to SCSI inquiry commands) to DMP's list of JBOD (physical disk) devices.

Array Support Libraries

For arrays that require more specialized handling, DMP's architecture provides for array-specific *array support libraries* (ASLs) for discovery and configuration and kernel mode *array policy modules* (APMs) that perform array-specific functions in the I/O path.

Figure 12 illustrates how ASLs and APMs fit into VxVM's configuration facilities and I/O path, with emphasis on the relationship to the `vxconfigd` configuration daemon.

they support are connected) can dramatically improve the speed with which a system starts up. Dialog 5 illustrates the sequence of VxVM commands for deactivating an unused ASL (the ASL remains installed and can be reactivated).

```
# vxddladm excludearray libname=libvxvpath.so
# vxdctl enable
```

Dialog 5: Deactivating an Unused ASL

When an ASL is deactivated in this way, the multi-path properties of the LUNs it controls do not change until VxVM discovery runs. During VxVM discovery, LUNs that had been controlled by a deactivated ASL are classified as generic disks. After the **vxddladm** command in Dialog 5 deactivates an ASL, the **vxdctl enable** command causes DMP discovery and reconstruction of its metanodes to reflect changes in device multi-path capabilities.

ASLs can be installed dynamically while VxVM is running. This makes it possible to add multi-path access control for new disk array models without stopping VxVM or rebooting the system. Installing an ASL does not automatically cause VxVM to recognize LUNs presented by new arrays, however. After ASL installation, the **vxdctl enable** VxVM command must be run to cause VxVM to discover new devices and their multi-path capabilities. Alternatively, if the locations of newly added devices are known, the **vxdisk scandisks** command can be issued with constraints to cause a (faster) partial device scan.

Array Policy Modules

Array Policy Modules (APMs) are dynamically loadable kernel modules invoked by the **vxdmp** driver to perform disk array-specific path selection and failover, error processing, and SCSI reservation and release. DMP includes default procedures for these common functions; installing an APM overrides the default procedure for all arrays whose array models refer to it.

As shipped by VERITAS, DMP APMs support generic active-active, active-passive, active-passive with group failover, in both single-host and multi-host configurations. Each array model includes a set of vectors that point to functions which implement policies such as:

- **I/O request routing**, using one of the six built-in policies discussed earlier (page 15)
- **Error handling**, including analysis, recovery, and DMP state changes. Built-in error handling policies include inquiry (the most common policy, described later), read-only path (for certain active-active array conditions such as EMC Symmetrix non-disruptive upgrade), and coordinated failover and failback for active-passive arrays in clusters
- **Get Path State**, for obtaining information about current path and device configuration for use in error handling and elsewhere
- **LUN group failover**, for active-passive arrays that support concurrent failover of en-

tire LUN groups triggered a single event

- **Explicit failover**, for arrays that support explicit failover functionality (page 6)
- **Failover path selection**, using first available path, primary path preferred, or other alternate path selection algorithms

DMP includes one or more default procedures for each of these policies. Custom APMs that implement array-specific procedures can be substituted by creating array models that vector to the procedures that implement custom functions.

DMP Device Discovery during System Operation

If a system's storage device configuration changes, for example because a device fails, or because additional disks or arrays are added, these must be discovered as well. Rebooting an operating system after a storage configuration change causes discovery, but rebooting is almost never desirable, especially for enterprise-class systems. UNIX operating systems therefore provide commands that an administrator can invoke to discover storage devices on demand. Table 1 lists the device discovery commands in each of the UNIX operating systems supported by DMP.

Operating System	Storage Device Discovery Commands
Solaris	devfsadm command performs subsystem scan, updates the device tree and loads drivers as necessary
AIX	cfgmgr command performs subsystem scan, updates the device tree and loads drivers as necessary
HPUX	Administrators should use the ioscan command to survey the old configuration, followed by the insf -e command to update the device tree and load drivers as necessary.
Linux	makedev command can be used to update the device tree, but I/O subsystem scan and driver loading are only done at boot time.

Table 1: UNIX Operating System Commands for Run-Time Storage Device Discovery

VxVM Device Discovery During Operation

Whenever an operating system rediscovers its storage configuration, VxVM must also discover any effects of the change on virtualization and multi-path access. Administrators can use one of two VxVM commands to cause rediscovery by the **vxconfigd** daemon:

- **vxctl enable**. This command causes **vxconfigd** to scan all storage devices and reconstruct DMP metanodes and other structures to reflect the current device configuration.
- **vxdisk scandisks**. This command may specify complete discovery, or it may be constrained to scan only newly added devices, or designated enclosures, array controllers or device address ranges. A limited scan can be considerably faster than a full one if a small number of changes have been made to a large storage configuration.

Both commands use the **vxconfigd** daemon to re-scan the storage configuration and update in-memory data structures to reflect changes since the previous scan. The daemon

uses ASL services to determine multi-path capabilities of newly-added devices. VxVM on-demand discovery does not interrupt system or application operation.

I/O Path Failover with DMP

Typically, the primary motivation for installing I/O path management software is to keep data accessible when an I/O path fails. A fundamental problem of I/O path management is distinguishing between failure of a path to a device and failure of the device itself. If a failed device is able to report its status (e.g., hard read error, unrecoverable positioning error, or write to a write-locked device), the determination is easy. More difficult to diagnose is a device that simply does not respond to an I/O request:

- Has the device failed?
- Has the path to the device failed, leaving the device unable to communicate the status of outstanding requests or accept new ones, even though it is functioning?
- Is the device simply too busy to respond within its timeout period?

When an I/O request fails, DMP must determine whether to re-route future I/O requests to alternate paths.

Path Failure Analysis

The `dmperrod` and `restored` daemons, both of which run as kernel threads, perform path failover and failback respectively. When either DMP or an operating system driver times out an I/O request to a multi-path device, DMP moves the request to its error handling queue. The `dmperrod` daemon diagnoses requests in the error handling queue by issuing one or more SCSI inquiry commands to the target device on the suspect path. (Most storage devices will respond to SCSI inquiry commands, even if they are unable to transfer data). If the daemon receives a valid response within the timeout interval, it concludes that the device was simply too busy to respond, and re-queues the request and any others added to the error queue in the interim, on the original path.

Path Failover

If the `dmperrod` daemon's SCSI inquiries fail within a timeout interval, it fails I/O over to an alternate path. For active-active arrays, path failover consists simply of adjusting the load balancing algorithm in effect to account for the failed path. For active-passive arrays, DMP initiates failover, either *implicitly*, by re-queuing outstanding I/O commands to an alternate path, or *explicitly* by invoking services of the APM that corresponds to the LUN being failed over.

Usage Note

On Solaris, AIX, and Linux systems, the **dmperrd** daemon starts immediately when the DMP driver, **vxdmp**, begins to execute at system startup. On HP-UX, **dmperrd** does not start until the VxVM volume management daemon, **vold**, executes its first **ioctl** command. Thus, on HP-UX platforms, **dmperrd** may not be observable immediately upon system startup.

DMP fails paths over on a device-by-device basis. Thus, if a shared resource, such as a storage network switch, fails, DMP executes the failure analysis and failover procedures described above for all devices affected by the failure. APMs for disk arrays that support LUN group failover can fail over entire groups of LUNs in response to a single event.

Path Failback

DMP's **restored** daemon runs every **dmp_restore_daemon_interval** seconds (300 by default) to check I/O path state. The **dmp_restore_daemon_policy** tunable can be used to specify one of four path checking policies:

- **check_all**. This policy causes the **restored** daemon to issue an inquiry command on every path to every device controlled by DMP, irrespective of the path's state. This policy provides timely notification of path failures, but the large number of inquiry commands it issues may impact application performance in systems with a large number of storage devices.
- **check_disabled**. This policy causes the **restored** daemon to issue inquiry commands to failed paths (but not administratively disabled ones). While it does not provide proactive notification of path failure, the low overhead of this policy makes it attractive for some data center operations.
- **check_periodic**. This policy causes the **restored** daemon to execute the **check_disabled** policy except during every **dmp_restore_daemon_cycles**th **dmp_restore_daemon_interval**, when it executes the **check_all** policy. For example, with a **dmp_restore_daemon_interval** of 300 seconds and a **dmp_restore_daemon_cycles** of 10, the **restored** daemon would issue inquiries to **FAILED** paths every 300 seconds (5 minutes), and to all devices on all paths every 3000 seconds (50 minutes). This policy represents a compromise between overhead and timely discovery of failed paths.
- **check_alternate**. This policy is similar to the **check_all** policy, except that the **restored** daemon stops issuing inquiry commands to a device when it finds two operational paths. This policy is useful with complex storage network topologies, where checking all paths might result in a large number of inquiries, possibly impacting application performance. Because it checks for at least one functional alternate path, this policy also protects against sudden path failure.

Administrators use the **vxdmpadm** utility to adjust the three tunables that control the **restored** daemon's behavior, **dmp_restore_daemon_interval**,

`dmp_restore_daemon_policy`, and, for the `check_periodic` policy only, `dmp_restore_daemon_cycles`.

DMP Configuration and Tuning Considerations

There are three key DMP parameters that administrators should understand and actively manage, because they can affect system performance and availability significantly. The values of these parameters are set differently for each platform implementation of VxVM:

- **Solaris.** The parameters are found in the `/kernel/drv/vxdmp.conf` file on Solaris platforms. The `vxdmpadm` command is used to set their values.
- **AIX.** The menu-based `smit` or the graphical `smitty` configuration management utilities are used to set the values of these parameters by selecting the `vxvm` option, navigating to the parameter to be changed, and setting its new value.
- **HP-UX.** Current values for DMP parameters can be determined by running the System Administration Manager (SAM) and selecting the **Kernel Configuration** and **Configuration Parameters** options. The parameters themselves are stored in the `/stand/system` file. Some parameter changes may require re-linking the operating system kernel, and rebooting the system using the new kernel.
- **Linux.** The parameters are found in the `/etc/vx/vxdmp_tunables` file. The `vxdmptune` utility is used to set their values.

The paragraphs that follow describe functions of the three key parameters.

DMP_FAILED_IO_THRESHOLD

The `dmp_failed_io_threshhold` parameter represents the amount of time beyond which DMP considers an I/O request failure to represent a storage device failure. As Figure 8 (page 12) suggests, the VxVM virtualization layer issues I/O requests to DMP metadevices. Before forwarding a request to the operating system SCSI driver DMP saves the current system time. If the SCSI driver signals that an I/O request has failed, DMP samples the time at the moment of failure notification, and computes how long the request was outstanding. If the request was outstanding longer than `dmp_failed_io_threshhold` seconds, DMP considers the device to have failed, and does not perform error recovery or path failover. If the request fails within `dmp_failed_io_threshhold` seconds, DMP considers path failure as a possible cause, and initiates the error analysis and recovery procedures described earlier.

By default, the `dmp_failed_io_threshhold` parameter is set to ten minutes. For non-redundant volumes, this is typically adequate; too low a value can result in I/O request failures that are actually due to transient storage network errors. For failure tolerant (mirrored) volumes, however, a lower value is usually appropriate, because the VxVM virtualization layer retries a failed I/O request on another of the volume's plexes. For failure tolerant volumes, therefore, the `dmp_failed_io_threshhold` parameter should typically be set to a few tens of seconds. Of course, appropriate settings depend on steady-

state storage network and device loading as well as individual disk or LUN performance characteristics.

The `dmp_failed_io_threshold` parameter is not used on the HP-UX platform. The HP-UX operating system provides an I/O request timing capability, called `pfifo` that causes the SCSI driver to abort I/O requests that remain outstanding for too long. DMP treats I/O requests timed out by the HP-UX SCSI driver identically to failed requests on other platforms for which the `dmp_failed_io_threshold` time is exceeded. The `pfifo` parameter can be set separately for each VxVM disk group using the `vxpfito` utility.

DMP_RETRY_COUNT

When a DMP I/O request to a SCSI driver fails within the `dmp_failed_io_threshold` interval, the `dmperrd` daemon begins recovery by issuing SCSI inquiry commands to the target device on the suspect path. The daemon issues as many as `dmp_retry_count` inquiries. The default value for `dmp_retry_count` is 5.⁷ To reduce failover times in storage networks with extensive multi-path connectivity, this value can be lowered. A value of 2 is usually appropriate for storage networks with two or more paths to each device.

Upper layers of the I/O software stack must ultimately interpret I/O failures. Success of a DMP inquiry command followed by repeated failure of an I/O request can occur for a variety of reasons, some of them application-related. For example, write commands to write-protected devices always fail immediately. The `dmperrd` daemon's SCSI inquiry will succeed, but the application's command will continue to fail. Behavior like this usually indicates an application or administrative error.

As another example, if a disk that is part of a non-redundant LUN is failing intermittently, an application I/O request to the LUN might fail, but the `dmperrd` SCSI inquiry to the LUN may succeed. In this case, the hardware failure must be detected and diagnosed by means external to VxVM (e.g., by the disk array's error reporting mechanisms) and acted upon.

DMP_PATHSWITCH_BLK_SHIFT

For LUNs configured to use the balanced path I/O policy (page 15), DMP divides the starting data address specified in each I/O request by the parameter `dmp_pathswitch_blk_shift` and discards the remainder. The quotient of the division modulo the number of active paths becomes an index to the active path on which the I/O request is issued.

The `dmp_pathswitch_blk_shift` parameter is system-wide; it applies to all storage devices using the balanced path load balancing policy. Its default value is one megabyte (2048 blocks) on all supported platforms, but can be changed to match application re-

⁷ ,On the HP-UX platform, the default value for `dmp_retry_count` is 30 because the HP-UX SCSI driver does fewer retries than those of other platforms.

quirements. For example, if an application reads large files sequentially using 2-megabyte requests, setting the value of `dmp_pathswitch_blks_shift` to 2 megabytes tends to alternate successive read requests among active paths. If the disk array recognizes sequential access patterns and reads ahead, application requests may be satisfied from data pre-read into cache.

An Additional AIX-Specific DMP Tuning Consideration

In addition to the three parameters that affect DMP availability and performance on all platforms, the `dmp_queue_depth` parameter is unique to the AIX platform. The `dmp_queue_depth` parameter limits the number of I/O requests to a single device that DMP will forward to the operating system driver in order to limit the time required to abort and redirect outstanding requests when a path fails.

While the `dmperrd` daemon is analyzing a possible path failure, application I/O requests may continue to arrive. If the `dmperrd` daemon ultimately determines that the path has failed, DMP must abort all I/O requests outstanding to the operating system driver and reissue them on alternate paths. The operating system driver may take a long time to abort a large number of I/O requests, so DMP *throttles*, or limits the number of I/O requests that it allows to be outstanding to the operating system driver to the value of the `dmp_queue_depth` parameter.

DMP relays only `dmp_queue_depth` requests for a single device to the operating system driver; it retains any additional ones in its own queue. Each time a request completes, the number outstanding drops below `dmp_queue_depth`. If there are additional requests for the device in DMP's queue, a DMP kernel thread issues another one to the operating system driver.

The default value of the `dmp_queue_depth` parameter is 32. As with other parameters, the value of `dmp_queue_depth` can be changed by running the AIX `smit` utility and selecting the `vxvm` subcommand.

Storage Network Hardware Settings

In addition to VxVM parameters, parameters that control the operation of host bus adapters and storage network switches can affect the performance and function of DMP. These parameters differ from vendor to vendor, but since similar components perform similar functions, conceptual similarity is found between HBAs and switches designed by different vendors. The paragraphs that follow describe HBA and storage network settings that may affect DMP operation and that are typically adjustable by system administrators.

Host Bus Adapter Settings

Host bus adapters (HBAs) send and receive user data, I/O requests, and control messages between host server memory and storage devices. To accomplish this, they transform data between host memory format and I/O bus or storage network format, and add protocol information (for transmission) or remove it (upon reception). The driver software that

controls HBAs is at the lowest level of the storage I/O stack (Figure 8 on page 12).

Because HBA drivers control access to I/O paths, they are often the point at which host servers first detect path failures. HBA designers build mechanisms into their drivers to help detect and analyze path failures. These mechanisms are typically controlled by adjustable parameters. The types of HBA parameters that affect DMP performance include:

- **Link-down timeouts.** Operating system driver I/O request timeouts occur either because a device is unable to respond to a host message in time, or because the path on which a message was sent has failed. To detect path failure, HBAs typically start a timer each time a message is received, and report link failure if the timer lapses without receipt of a message. Most HBAs provide for user adjustment of a *link-down timeout* period. The link-down timeout interval should approximately equal DMP's `dmp_failed_io_threshold` parameter value. If the link-down timeout period is significantly shorter than `dmp_failed_io_threshold`, DMP will not detect path failures as quickly it should. If the HBA link-down timeout is significantly longer than `dmp_failed_io_threshold`, DMP may time I/O requests out when in fact nothing is wrong.
- **Link retry count.** Links that are inherently noisy may experience more frequent transmission errors than links in less noisy environments. Most HBAs include some type of adjustable link retry count that can be adjusted upward if necessary to prevent premature failovers on links that experience periodic noise bursts.

Storage Network Switch Settings

Storage network switches also expose settable configuration parameters that can affect the operation of DMP. The two most common ones are:

- **Interoperability mode.** Switch and director vendors often build proprietary enhancements to standard protocols into their products. While these enhancements are useful in homogeneous networks (those that include only one vendor's switches), they may not function correctly or optimally in heterogeneous networks, or with disks and disk arrays that have not been certified for operation with the switch vendor's products. Switches typically have an interoperability mode in which they adhere strictly to standard protocols for I/O and for inter-switch communications. Since DMP assumes standards-compliant storage device behavior, it is usually advisable to operate a storage network in this interoperability mode.
- **Buffer credits.** Buffer credits are a throttling mechanism used by HBAs and storage devices to avoid being swamped by incoming message and user data frames. According to Fibre Channel protocols, an originator is only permitted to send a frame to a receiver when the receiver has granted it a buffer credit. For long paths, particularly those with multiple "hops" between intermediate switches, buffer credit shortages can increase latency (the elapsed time for sending a message from originator to receiver) well beyond what would be expected given the bandwidth and loading of the link between the two. Alternate paths with different numbers of "hops" might have significantly different average latencies, and if timeout parameters are set for the shorter

path, “false” timeouts might occur frequently on the longer one. The best way to avoid timeouts caused by a lack of buffer credits is to increase the number of buffer credits that HBAs and disk array ports grant for multi-hop links. If this cannot be done, DMP and operating system driver timeouts should be raised to accommodate typical latencies actually encountered on longer paths.

Using DMP with other Path Managers

Path managers that are functionally similar to DMP are available from some disk array and system vendors. There are three principal differences between DMP and the path managers typically supplied by hardware vendors:

- **Heterogeneity.** Typically, hardware vendors’ path managers are optimized to support the vendor’s own products, sometimes to the exclusion of other vendors’ storage and systems. VxVM on the other hand is inherently heterogeneous, both with respect to platforms, storage devices, and in its ability to intermix different types of storage in a single virtual device configuration.
- **Integration.** VxVM is more closely integrated with the VxFS file system and with major database managers than is typical for hardware vendor-supplied virtualization and path management software. Integration with adjacent layers in the I/O software stack tends to minimize administrative complexity, and therefore, cost.
- **Cost.** Vendor path managers are typically extra-cost adjuncts to disk arrays, whereas DMP is an integral part of the VERITAS Storage Foundation software, the purchase of which is usually amply justified on other grounds.

For a variety of reasons, hardware vendor-supplied path managers are sometimes present on systems where DMP is also installed. Both approaches have their own benefits and limitations. To manage systems in which two path managers are present, storage administrators should appreciate how DMP and other path managers interact with each other.

Path Managers in the Storage I/O Stack

Figure 13 represents the main functions performed in a UNIX storage I/O software stack, both with and without virtualization and path management. File systems and database managers configured to use “raw” devices make their I/O requests directly to operating system drivers that interact with disks and LUNs via host bus adapters. UNIX operating system drivers consist of:

- **A SCSI layer (*sd*)** that transforms file system and database manager requests into the format required by the storage device interface (e.g., Command Data Blocks, or CDBs, for SCSI protocol)
- **A host bus adapter (HBA) layer** that interacts with HBAs to set up the transfer of SCSI CDBs, device responses, and data between host server and storage device.

Path ❶ in Figure 13 represents the direct software connection between file system and storage device. In this scenario, the file system makes I/O requests to the operating system’s SCSI driver, which reformats them and passes them to an HBA driver. HBA driv-

ers treat I/O requests and device responses as messages which they send between source and destination without interpretation (a direct memory access, or DMA, mechanism is typically used for data transfer).

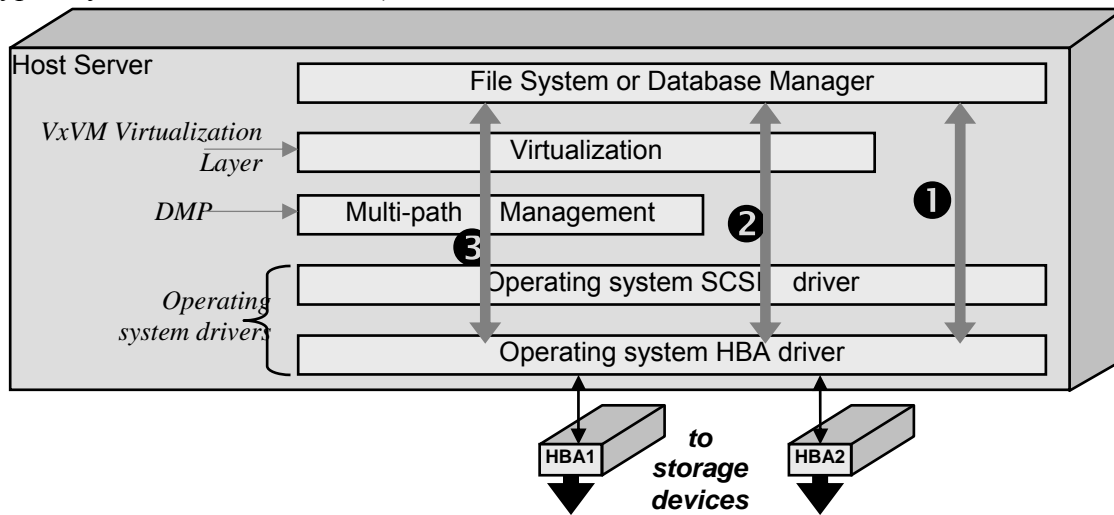


Figure 13: Functions Performed at Different Layers of the UNIX I/O Software Stack

Host-Based Virtualization Software

Host-based storage virtualization software like VxVM adds a layer to the stack. The virtualization layer manages collections of storage devices which it represents to file systems and database managers as disk-like *virtual volumes* with improved data availability, I/O performance, and functionality. Path ② in Figure 13 represents the software stack traversed by file system I/O requests to virtual volumes. In this scenario, file system I/O requests to virtual volumes are fielded by a virtualization layer such as VxVM. The virtualization layer creates equivalent requests to physical disks or LUNs, and issues them to operating system drivers.

For example, VxVM transforms each application write request to a mirrored volume into a write request to each of the volume's mirrors, and each read request into a request to *one* of the volume's mirrors. More complex transformations also occur, as for example, a write request to a RAID volume which VxVM transforms into a series of read and write requests interspersed by computations, all of which collectively maintain parity consistency while user data is updated.

Path Managers

As demonstrated earlier, multi-path management further enhances data availability and I/O performance. Path ③ in Figure 13 represents an I/O software stack that includes a path manager such as DMP. In this scenario, the file system makes I/O requests to volumes. The virtualization layer transforms file system requests, and makes its I/O requests to metadevices presented by the path manager. The path manager selects an I/O path for each request according to the load balancing policy in effect, and issues the request to the operating system SCSI driver. When host-based virtualization and path management are

both in use, file system I/O requests are transformed twice before actually being issued to storage devices:

- Each file system request to a virtual volume is transformed into one or more requests to metadevices that represent physical disks or LUNs
- Each virtualization layer request to a disk or LUN metadvice is transformed into an operating system driver layer request to a device on a specific access path

No value is gained by performing the second transformation more than once. Historically, it has been regarded as poor practice to configure more than one path manager for the same device. Generally, path managers are unaware of each other, and therefore do not coordinate operating system I/O requests, or more importantly, error recovery actions.

Typically, hardware vendor-supplied path managers can be enabled or disabled either on a per-device basis or system-wide. By default, VxVM manages all devices that it recognizes as disks or LUNs, including controlling access paths, but can be disabled selectively, as Dialog 6 illustrates.

```
# mkt1 # vxdiskadm
Volume Manager Support Operations
Menu: VolumeManager/Disk
 1      Add or initialize one or more disks
 2      Encapsulate one or more disks
 3      Remove a disk
 4      Remove a disk for replacement
 5      Replace a failed or removed disk
 6      Mirror volumes on a disk
 7      Move volumes from a disk
 8      Enable access to (import) a disk group
 9      Remove access to (deport) a disk group
10     Enable (online) a disk device
11     Disable (offline) a disk device
12     Mark a disk as a spare for a disk group
13     Turn off the spare flag on a disk
14     Unrelocate subdisks back to a disk
15     Exclude a disk from hot-relocation use
16     Make a disk available for hot-relocation use
17     Prevent multipathing/Suppress devices from VxVM's view
18     Allow multipathing/Unsuppress devices from VxVM's view
19     List currently suppressed/non-multipathed devices
20     Change the disk naming scheme
21     Get the newly connected/zoned disks in VxVM view
22     Change/Display the default disk layouts
23     Mark a disk as allocator-reserved for a disk group
24     Turn off the allocator-reserved flag on a disk
list   List disk information
?      Display help about menu
??     Display help about the menuing system
q      Exit from menus
Select an operation to perform: 17
```

Dialog 6: Disabling DMP for Selected Devices

How DMP Coexists with Other Path Managers

Because all path managers perform essentially the same function, all behave in essentially the same way. They create pseudo-device nodes for the devices they control, and redirect file system or virtualization layer I/O requests made to these pseudo-devices along one of the paths to an actual device.

When a UNIX system starts up, it creates entries in its device tree for each block access storage device (disk or LUN) it discovers. Device tree entries for multi-path devices are referred to as *sub-paths*. Each storage device sub-path has a `DDI_NT_BLOCK_WWN` attribute indicating that it represents a block access storage device. Some path managers *suppress* access paths by turning this indicator off, thus preventing other software from directly accessing devices they control. For example, DMP's device discovery layer treats suppressed paths as though they did not represent storage devices.

Path-Suppressing Path Managers

Path managers that suppress device paths do so in one of two ways:

- They reset the `DDI_NT_BLOCK_WWN` indicator for all but one sub-path to each device they control. For example, if `c1t1d1` and `c2t1d1` represent two sub-paths to a device, a path manager might suppress `c1t1d1` and leave the `DDI_NT_BLOCK_WWN` indicator for `c2t1d1` turned on. Path managers that do this include EMC Corporation's *Automatic Transparent Failover* (ATF) and LSI Logic's *Redundant Disk Array Controller* (RDAC), both of which use unsuppressed paths as metanodes for the devices they represent.
- They replace device tree entries that represent paths to devices they control with their own metanodes (with the `DDI_NT_BLOCK_WWN` indicator on). These path managers typically name their metanodes in some distinctive form (e.g., `cXtWWWdXsX`, where `WWW` represents a device worldwide name), rather than the common `cXtXdXsX` operating system form. They also enable a `DDI_NT_FABRIC` property, which causes devices represented by their metanodes to be treated as storage network fabric devices. Sun Microsystems' MPxIO path manager operates in this way.

In both of these cases, path managers effectively suppress all but one path to each device under their control. DMP can create its own metanodes (as illustrated in Figure 9 on page 14) linked to a path-suppressing path manager's pseudo-devices, and co-exist with the other path manager for most purposes, but because each pseudo-device appears to DMP as a single-path device, DMP performs no useful function.

Path Managers That Do Not Suppress Device Paths

Other path managers do not suppress sub-paths to the devices they control, but leave them visible to other software, including DMP. Non-path suppressing path managers treat sub-paths for the devices they control in one of two ways:

- They leave the sub-paths unmodified, and add their own metanodes to the device tree. With such a path manager, a LUN accessible on two paths is represented by three de-

vice tree entries—two for the operating system-discovered sub-paths and one for the path manager’s metanode. Metanodes created by these path managers have characteristic naming patterns that differ from the typical operating system `cxtdxsx` pattern (e.g., Sun Microsystems’ *Alternate Pathing* (AP) driver metanode names follow the pattern `mcxtxsx`. IBM Corporation’s Subsystem Device Driver (SDD on the Solaris platform and VPATH on AIX) metanodes names take the form `vpathx`).

- They leave sub-paths intact, and insert their metanodes in a separate file system directory. Metanodes created by these path managers also have characteristic naming patterns. Vendors of these path managers historically have not provided APIs that would allow DMP to determine relationships between metanodes and sub-paths, so DMP has historically had difficulty coexisting with them, although this situation is changing. EMC Corporation’s PowerPath behaves in this fashion.

Starting with Version 4.1 of VxVM, DMP can co-exist with non-suppressing path managers, essentially by locating their metanodes, identifying them with the corresponding operating system paths, and suppressing all but one of the paths. DMP creates its metanode pointing to the single unsuppressed path.

Table 2 lists the path suppression characteristics of common path managers.

Path Management Driver	Vendor	Type	Arrays	Platforms
ATF (Automatic Transparent Failover)	EMC Corporation	Path suppressing	FC4700	Solaris
RDAC (Redundant Disk Array Controller)	IBM Corporation	Path suppressing	Sonoma..... FAST.....	Solaris AIX
STMS	Sun Microsystems	Path suppressing	T3, T4,	Solaris
SDD (called VPATH on AIX platform)	IBM Corporation	Non-path suppressing	ESS (Shark)	Solaris
PowerPath (Native Mode)	EMC Corporation	Non-path suppressing	Symmetrix CLARiion	Solaris HP-UX Linux
PowerPath (Pseudo mode)	EMC Corporation	Non-path suppressing	Symmetrix CLARiion	Solaris

Table 2: Path Suppression Characteristics of Common Path Managers

DMP and Foreign Devices

Unless somehow prevented from doing so, DMP would discover both the sub-paths and metanodes of non-path suppressing path managers. DMP and the other path manager might both attempt to manage access to the same devices, with obvious conflicts. To avoid this, Version 4 of the Storage Foundation introduced the concept of *foreign* devices. The `vxddladm addforeign` command is used to declare a device to be foreign. DMP does not control path access to foreign devices, but the VxVM virtualization layer can still incorporate them in disk groups and use them as volume components.

DMP and EMC's PowerPath

EMC Corporation's PowerPath software is frequently-installed on systems together with DMP. PowerPath software operates either in *pseudo device mode* (also called *emcpower mode*) or in *native device mode*, both of which are non-path suppressing. In both operating modes, replaces operating system SCSI driver call vectors with calls to its own path management routines.

PowerPath Pseudo Mode

In pseudo device mode, PowerPath creates an **emcpowerx** pseudo-device for each device it controls. The **emcpowerx** device is conceptually similar to a DMP metanode. **Error! Reference source not found.** shows the key properties of a PowerPath pseudo-device as displayed by the EMC **Powermt** utility.

Powermt display

```
Pseudo name=emcpower41a
Symmetrix ID=000184500052
Logical device ID=0003
state=alive; policy=SymmOpt; priority=0; queued-I/Os=0
=====
----- Host ----- - Stor - -- I/O Path - -- Stats ---
### HW Path          I/O Paths   Interf.   Mode    State  Q-I/Os  Errors
=====
3084 pci@bd/SUNW,qlc@1    c16t50060482BF CFD502d0s0 FA  3aA   active  alive    0
0
3085 pci@bd/SUNW,qlc@1    c17t50060482BF CFD51Dd0s0 FA  14bA  active  alive    0
0
```

Dialog 7: PowerPath Pseudo-Device Attributes

Figure 14 illustrates the three possible paths through the I/O software stack in a system with both DMP and PowerPath in pseudo device mode installed.

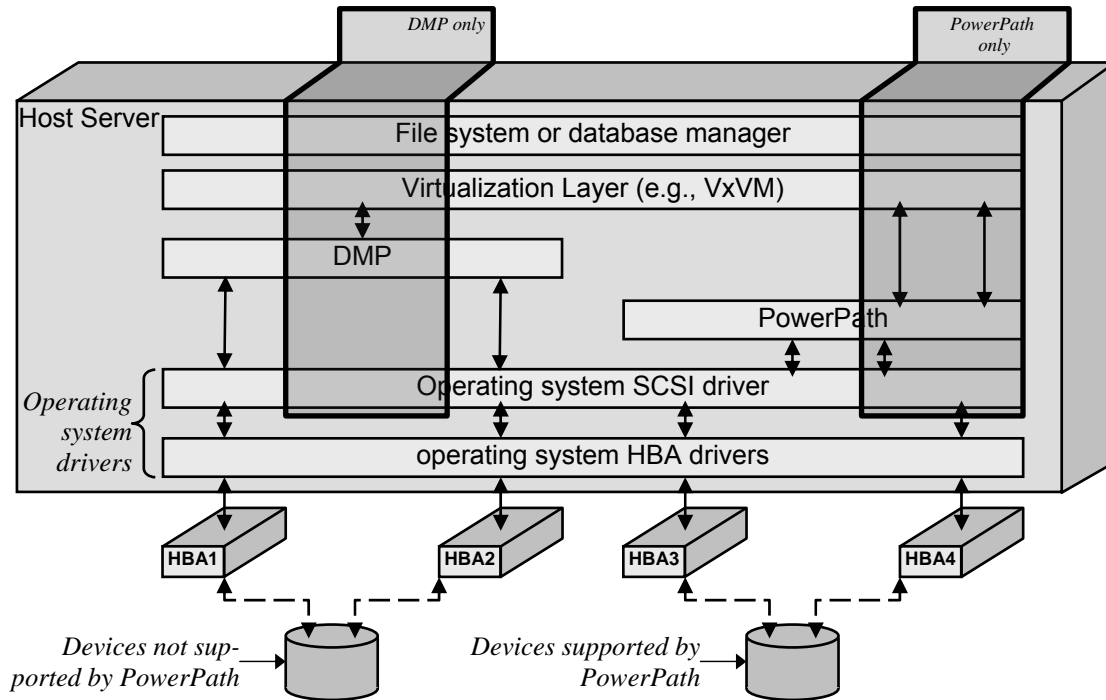


Figure 14: DMP with EMC PowerPath in Pseudo Device Mode

PowerPath is non-suppressing; it leaves operating system nodes for the devices it controls visible in the device tree. DMP does not recognize PowerPath pseudo-devices, but it does recognize sub-paths to devices that PowerPath controls, and will create metanodes and attempt to control them (center shaded area in Figure 14), unless it is prevented from doing so.

EMC Corporation delivers a script called `powervxvm` that makes it possible to use `emcpower` devices with VxVM Version 3.5 and newer versions. Starting with Version 4 of VxVM, the `powervxvm` script declares PowerPath-controlled devices to be foreign. In both cases, DMP does not claim the sub-paths of the `emcpower` devices, so limited coexistence, illustrated by the rightmost shaded area in Figure 14, is possible. When coexisting with PowerPath, VxVM Version 4 and earlier versions cannot make `emcpower` devices portable between different platforms, nor can it use them as shared storage in Storage Foundation for Real Application Clusters (Oracle RAC) configurations. With Version 4.1 of VxVM, these restrictions are relaxed, and `emcpower` devices can be used as RAC cluster storage.

As the leftmost shaded area in Figure 14 suggests, DMP can control path access to storage devices not managed by PowerPath (e.g., devices that PowerPath does not support). PowerPath discovery ignores these devices, so their sub-paths are visible to DMP discovery, as they would be in a system without PowerPath installed.

PowerPath *Native-Mode*

Primarily for compatibility with existing applications and scripts, PowerPath also supports storage devices in non-suppressing *native device mode* on certain platforms (Table

2, page 33). Disks and LUNs controlled by PowerPath in native device mode are accessed by their familiar operating system names, so applications and scripts work without modification.

When operating in native device mode, PowerPath does not create pseudo-device nodes, but it does control path access to its devices. Native mode PowerPath directs each I/O request to the sub-path indicated by its load balancing policy, no matter which sub-path originally received the request. If DMP were to control a PowerPath native mode device, as the center shaded area in Figure 15 suggests, VxVM I/O virtualization layer requests might reach target devices on a different path than the one on which it was issued.

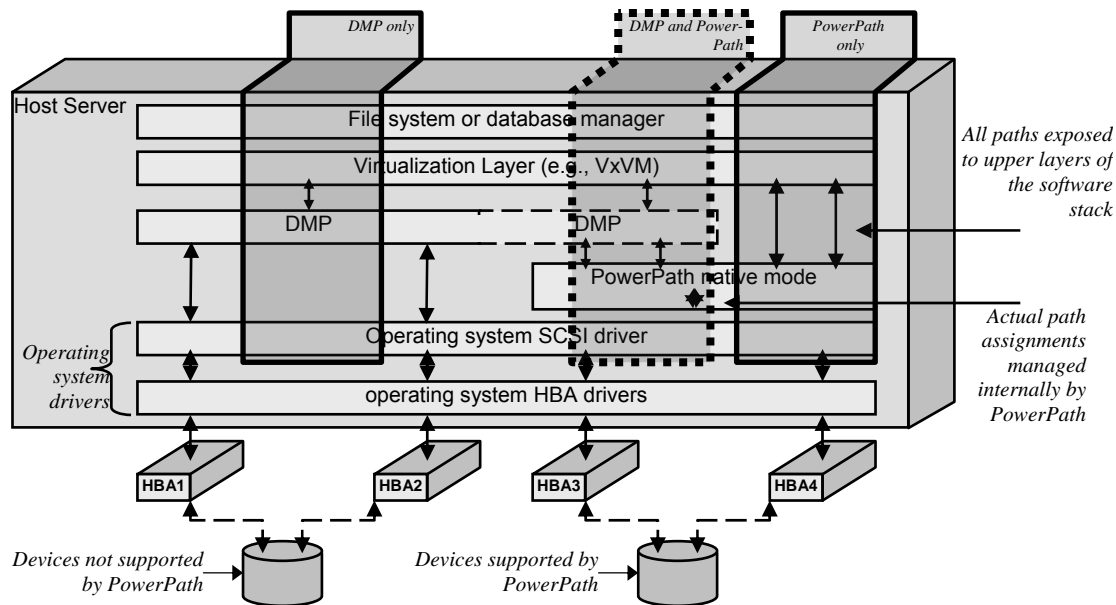


Figure 15: DMP with EMC PowerPath in Native Mode

If PowerPath is the only path manager in the stack, its re-routing of I/O requests is immaterial to the virtualization layer. But if DMP were also in the stack, it, too, would select a path for each I/O request. Native device mode PowerPath would re-route DMP's I/O requests, possibly affecting I/O performance adversely. Moreover, DMP and PowerPath error recovery are likely to conflict, with unpredictable results.

DMP and native mode PowerPath have been used successfully in cluster applications. VERITAS clusters use SCSI persistent group reservations to prevent so-called *split brain* behavior when the connection between two servers breaks. DMP propagates reservation requests on all paths to a device, but pseudo-device mode PowerPath does not (this is why emcpower devices are not usable as RAC shared storage prior to VxVM Version 4.1). With native mode PowerPath, however, all device paths are visible to DMP, so it can propagate device reservation requests on all paths. While this mode of operation has been used successfully, the possibility of the two path managers' error recovery algorithms interfering destructively with each other remains. Users with applications such as RAC, that require persistent group reservations are therefore advised to upgrade to the newest VxVM version promptly.

Path managers may have configuration requirements that are incompatible with DMP. For example, DMP requires that active-passive EMC arrays (e.g., Clariion Cx600) be configured for implicit failover, whereas PowerPath requires that they be configured for explicit failover. If a Cx600 array is configured for control by PowerPath, DMP cannot control its LUNs correctly, even if PowerPath is disabled.

Other I/O Path Management Software—STMS

I/O path management software is also available from other system and storage vendors. Sun Microsystems offers a path manager called *Storage Traffic Manager Software* (STMS) that supports its own disk arrays on major enterprise UNIX, Linux, and Microsoft Windows server operating systems.

STMS creates pseudo-devices called *Storage Traffic Manager Software devices* (STMS devices). STMS devices are conceptually similar to DMP metadevices; each one represents a LUN or disk along with all the access paths to it. STMS device names have the general form **Cx^twwn^dx^sx**, where each **x** represents a number assigned by STMS, and **wwn** represents the device's Fibre Channel worldwide name. STMS uses a policy similar to DMP's round robin policy (page 16) to balance I/O loads.

STMS can be enabled or disabled when a system starts. If enabled, it necessarily controls all the storage devices it supports (but ignores unsupported devices). To enable STMS, an administrator adds the line: `'mpxio-disable="no";'` to the `/kernel/drv/scsi_vhci.conf` file. To disable STMS, the `"no"` is replaced with a `"yes"` in the expression. After the line is changed, the system must be rebooted for the change to take effect.

Figure 16 illustrates the I/O stack in a system in which DMP and STMS are both installed. STMS is path-suppressing (Table 2, page 33), so when it is enabled, other software "sees" only single-path STMS devices for disks and LUNs supported by STMS. Thus, DMP does not actually manage access paths to STMS devices. DMP can, however, manage access paths for devices not supported by STMS, as suggested by the left side of Figure 16.

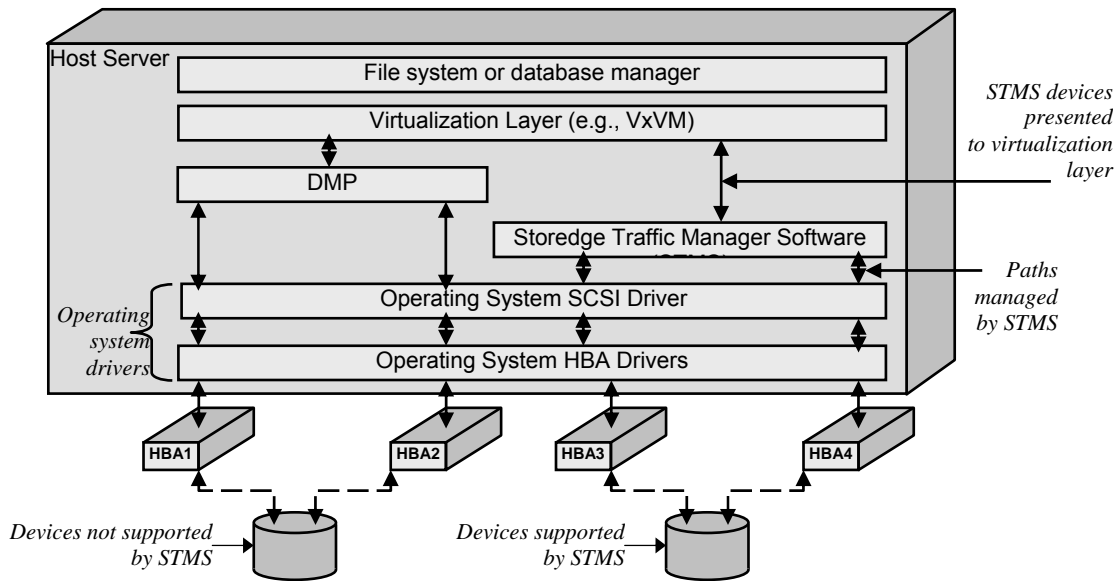


Figure 16: STMS-DMP Comparison

Other I/O Path Management Software—SDD

IBM Corporation offers *Subsystem Device Driver* (SDD) software that manages access paths for its own disk arrays (ESS, DS8000, DS6000) as well as SAN Volume Controller switch-based virtualizers. SDD software is available for major enterprise UNIX, Linux, Netware, and Windows server operating systems. Figure 17 compares SDD with DMP in a system in which both are installed.

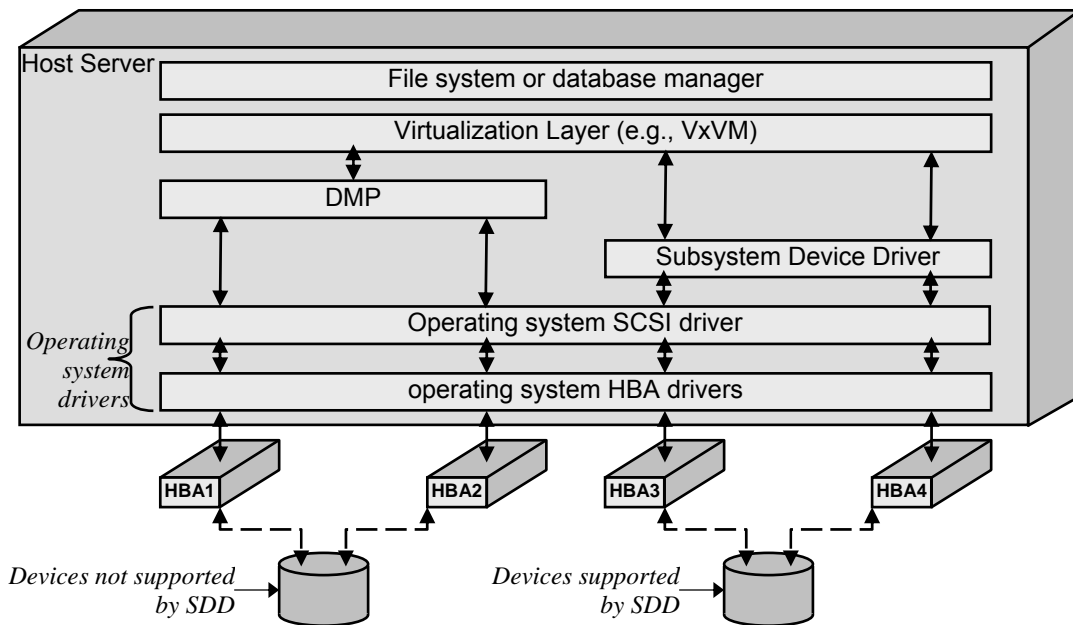


Figure 17: DMP and IBM's Subsystem Device Driver

SDD is non-path suppressing, so As Figure 17 suggests, higher layers in the I/O software

stack “see” all access paths to the devices it controls. By default, SDD manages path access to devices it supports, but unlike STMS, SDD does not necessarily control all of the devices it supports. In a system with SDD installed, DMP can manage path access to devices not supported by SDD, as well as to SDD-supported devices for which SDD has been disabled. An administrator can disable SDD by disabling its ASL and rerunning DMP discovery using a command sequence similar to that shown in Dialog 5 (page 21).

Conclusion

Effective management of multiple access paths to storage devices keeps data available to applications when storage network components fail. For disk arrays that support concurrent access to LUNs on multiple paths, balancing I/O load across two or more paths can improve I/O performance. To be effective, path management must be automatic, and load balancing must be consistent with data center policy.

The Dynamic Multipathing (DMP) feature of the VERITAS Storage Foundation automates both failover of storage devices to alternate paths when primary ones fail and balancing of I/O load across available paths according to any of several policies that can be set to achieve a variety of I/O performance goals.

LUNs may support concurrent access through multiple disk array controllers (active-active), or through only a single controller (active-passive). Active-passive disk arrays may support concurrent access on one or more primary paths (active-passive concurrent). In active-passive arrays, failover may be implicit (caused by issuing I/O commands on secondary paths) or explicit (caused by array-specific commands). Disk arrays may fail LUNs over from primary paths to secondary ones individually or in groups.

Major system and storage vendors offer path management software, usually to manage access paths for their own products. Vendor-supplied path management software can co-exist with DMP in a system, but configurations in which both attempt to manage the same devices should generally be avoided. Typically, DMP and vendor-supplied path management software should manage path access for different sets of devices.

DMP can control more different types of devices than typical vendor-supplied software. Additionally, it implements a more comprehensive set of I/O load balancing policies. In general, DMP specifications are technically superior to those of most vendor-supplied path managers, but care should be used in removing any path management software from a system, because unpredictable results may occur.



VERITAS Software Corporation
Corporate Headquarters
350 Ellis Street
Mountain View, CA 94043
650-527-8000 or 866-837-4827

For additional information about VERITAS Software, its products, VERITAS Architect Network, or the location of an office near you, please call our corporate headquarters or visit our Web site at

© 2005 VERITAS Software Corporation. All rights reserved. VERITAS, the VERITAS Logo, VERITAS Storage Foundation, and FlashSnap are trademarks or registered trademarks of VERITAS Software Corporation or its affiliates in the U.S. and other countries. Other names may be trademarks of their respective owners.