Management of Highly Available Storage Services

Danny B. Gross
Unix Administrator
Motorola, Inc
6501 William Cannon West
Austin, Texas 78735
danny.gross@motorola.com
(512) 895-4825
FAX (512) 895-4933

Chris Roberson
Storage Technical Lead
Hewlett Packard, Inc.
12401 Research Boulevard, Suite 200
Austin, Texas 78759
Chris_Roberson@hp.com
(512) 257-5721
FAX(512) 257-5791

05/09/01

Abstract

The Storage Area Network provides many benefits to the Highly Available environment. Disk space management on a large storage array can provide unparalleled opportunity for increases in performance and availability. However, correctly allocating disks, physical volume links, and taking advantage of volume striping can be daunting tasks for the most experienced administrator. This session provides a case study of a SAN environment in support of a Semiconductor Manufacturing operation. From this case study is developed a systems administrator's perspective of how the XP256 operates, and how to correctly identify and allocate disk space for various applications. Logical volume creation techniques are presented, including how to use physical volume groups and links to stripe and provide path redundancy. Also presented are troubleshooting techniques using real-world problems.

Storage Arrays, The SAN, and their role in the HA environment

To say that the Storage Area Network (SAN) provides many benefits to the Highly Available (HA) environment would be an understatement. In fact, storage is at the very center of the universe in any application. In an HA environment, the ability to configure, share, and failover applications storage rapidly and easily amongst many hosts while maintaining a high level of performance is the difference between prosperity and bankruptcy. For organizations striving to achieve even remotely close to the 5-Nines availability concept, there is no option for the use of storage arrays in a SAN environment.

The SAN and large storage arrays make it possible to support once unimaginable claims in even very complex HA environments. The ability to restart an entire semiconductor manufacturing control system within 20 minutes from a single host by a

single person is an incredible achievement. When one considers that the reboot of a single server within that environment can take upwards of 25 minutes, the claim is astounding. Without the storage array, the claim is next to impossible. Rapid applications qualification and deployment, cheap extension of HA components to less critical or non-production systems, and even live cross-campus datacenter relocation are realistic due to the SAN. Yet, for all its benefit, there is a cost in the complexity, relative youth, and architectural impact of storage services management.

By itself, the act of consolidating disk requirements from multiple hosts on large storage frames is complex enough. When the individual intricacies of storage arrays, Fibre Channel switching and looping technologies, incomplete standards and vendor claims of supremacy are factored in, the act of consolidation can be overwhelming. The technology is still very young, and every vendor has its own view of the SAN universe. It is both difficult and frustrating to determine and sell the solution that is correct for an environment using the typical eighteen-month vision. Adding to the mire is the fact that a minor component failure, performance issue or misconfiguration of a disk in consolidated storage can have a huge ripple effect across many hosts, and can cripple an entire environment. The youth of the technology has a tendency to leave the system administrator on the site as the most technically qualified SAN engineer in the industry during a problem. The HA architecture itself can further complicate the issue.

In designing HA frameworks, most of us have a tendency to buy into the popular technologies for clustering and management. Unfortunately, the industry is inundated with such tools that demand vendor-specific storage, network, and other technologies. In a SAN environment, this is certainly supportable. However, it is easy to imagine how complicated an environment can become by maintaining a narrow, project-based focus on HA. As time goes on, the environment becomes a mix of technologies and approaches, cluttered with the carcasses of outdated proprietary systems that cannot be upgraded but must be maintained. Such an environment built on a SAN is complicated, expensive, and extremely difficult to support. The key to seeing through all of the smoke and confusion and building success is to simplify the administrator's view of the environment, and architect flexibility everywhere. Once such a view is built, the management of storage services is reduced to standard Logical Volume Management procedures.

Building a simplified view

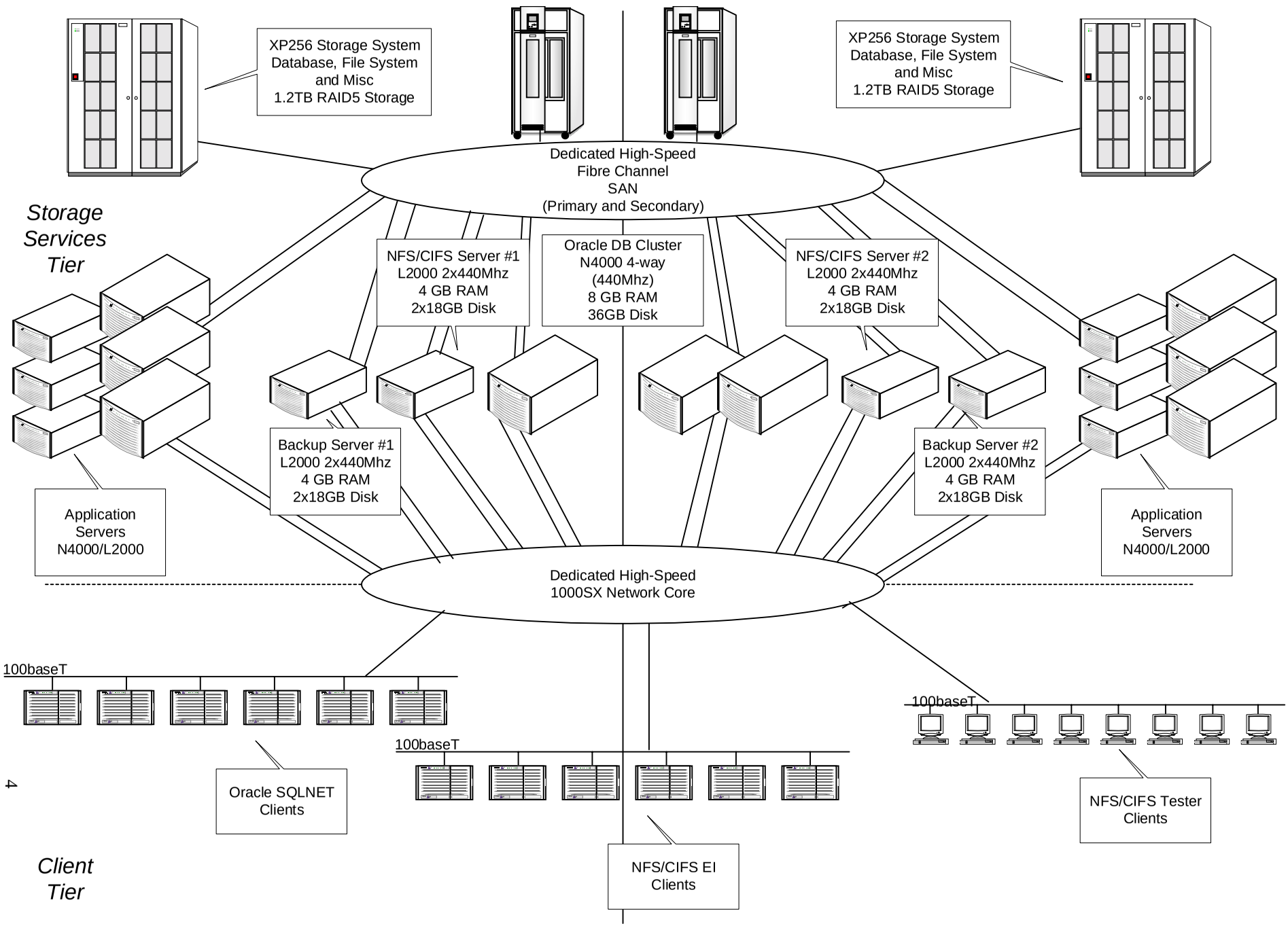
In its simplest sense, any environment has a management function, a storage service, a database service, and an application service. In a typical environment, simplification involved the "one application – one server" approach, with a failover system set aside. In this approach, all four components reside on each server. As more applications are built, more servers supply all four functions. This approach has its place, but complexity and performance problems proliferate rapidly. Truly, if we step back and look at the whole environment with an infrastructure-centric view, we would find that it makes sense to organize the environment in the following way:

- Separate the database from the application into a cluster of servers designed to do nothing but run databases.
- Separate the NFS server from the application and the databases into its own cluster of servers designed to do nothing but serve storage to all platforms.
- Design a way to centralize the management of all applications from a central point.
- Connect only Enterprise Storage, Enterprise-class backup, SAN switches, and Enterprise-class systems in a SAN.
- Store only operating system and transient data stored in the root volume group on every system. Serve all other storage in some fashion.
- Serve all non-OS and non-transient data via NFS to servers not connected directly to the SAN.
- Simplify NIS. Avoid the use of the automounter on SAN-connected servers, and keep the domain simple for the sake of the Storage servers.
- Use a Volume Manager (Veritas or LVM) on SAN-connected servers.
- Centralize and standardize the allocation of all storage on the SAN. Here is where the greatest flexibility is built into the infrastructure for HA purposes -- it determines which hosts are capable of activating a volume group, and ensures that multiple volume groups can be activated on a single host without conflict with others.

An organization that builds the infrastructure framework in this direction will find that the application becomes a very lightweight subset of its previous form. It also now makes sense to modify the general HA approach to an N+1 configuration, because the application server is capable of running multiple applications with increased performance. Performance tuning and systems load planning become much simpler as each framework component is organized separately from the other. Database systems can tune out NFS, Buffer Cache, and other parameters without fear of impacting their clients. Applications servers no longer need the Shared Memory components, asynchronous I/O, and NFS server components. Every disk served by the Storage Array is treated with identical priority. It is with this view that Motorola designed and built its MOS11 Manufacturing System.

MOS11 - an example of a simplified view

The following picture is an overview of the storage architecture of Motorola's MOS11 manufacturing facility.


XP256 Storage System
Database, File System
and Misc
1.2TB RAID5 Storage
XP256 Storage System
Database, File System
and Misc
1.2TB RAID5 Storage
Dedicated High-Speed
Fibre Channel
SAN
(Primary and Secondary)
Storage
Services
Tier
NFS/CIFS Server #1
L2000 2x440Mhz
4 GB RAM
2x18GB Disk
Oracle DB Cluster
N4000 4-way
(440Mhz)
8 GB RAM
36GB Disk
NFS/CIFS Server #2
L2000 2x440Mhz
4 GB RAM
2x18GB Disk
Backup Server #1
L2000 2x440Mhz
4 GB RAM
2x18GB Disk
Backup Server #2
L2000 2x440Mhz
4 GB RAM
2x18GB Disk
Application
Servers
N4000/L2000
Application
Servers
N4000/L2000
Dedicated High-Speed
1000SX Network Core
100baseT
100baseT
100baseT
Oracle SQLNET
Clients
NFS/CIFS EI
Clients
NFS/CIFS Tester
Clients
Client
Tier

Management of the Storage Services Component

The key to successful Storage Management is simplification and flexibility. If we must view each disk in an array with differing importance, the job of disk allocation becomes a complex nightmare, and we can quickly paint ourselves into a corner with regards to flexibility. If we view each disk in the array with identical importance, allocation becomes a simple, rapid and routine exercise. In order for us to see the truth in this, we should consider the internal workings of the HP XP256 array, and tools to help us organize the disks from both an array and host perspective.

An Internal view of the Hewlett-Packard Surestore E Disk Array XP256

The Hewlett-Packard Surestore E Disk Array XP256 (XP256) provides high level performance, availability, and reliability. The following paragraphs discuss three elements of the XP256 architecture: backend connectivity to disk, cache and bus architecture, and client host interface connectivity.

Backend Connectivity

The XP256 backend utilizes SCSI technology to attach the disk drives to the main system buses. The XP256 utilizes dual active FWD SCSI (20MB/sec) buses to each disk drive. Each disk drive supports dual ports providing the capability of accepting simultaneous I/O request. Since both ports work in tandem the dual buses provide additional I/O bandwidth under normal conditions, but in cases where one path fails the operational bus continues to provide access to the data disk although suffering from degraded performance.

The XP256 supports a high speed 15GB drive for high performance configurations and 36GB, 47GB, and 73GB drive for large capacity configurations. The drives can be combined in a single storage unit to provide a balanced configuration supporting performance and capacity. The dual bearing used in all XP256 drives allows the 15GB drive to achieve 12,000 RPMS. The dual bearings not only provide the highest rotational spin available, but also increase the Mean Time Between Failure (MTBF) from a typical 1.0 million hours to 2.5 million hours. The higher MTBF decreases support cost since fewer drives need replacement. For data protection the XP256 supports RAID1 (Mirroring) or RAID5 (Parity Striped).

On the XP256 a RAID1 disk configuration is used for large sequential read environments (Decision Support Systems) for high performance. RAID5 configurations are typically used in random read/write environments (On-line Transaction Processing) providing an additional 25% of usable disk capacity for only a 3% performance degradation over RAID1. The RAID5 configuration provides substantial savings in OLTP scenarios.

The XP256 Array Control Processors (ACP) tie together the back-end infrastructure of the disk array. Each ACP utilizes redundant processors with I/O fail-over capability in cases of chip level failures on an ACP. The ACP's DRR circuitry supports hardware

level RAID5/RAID1 allowing back-end parity calculations at disk connections. Each ACP provides four FWD SCSI channels dual-connected to one of four disk drives. The group of four drives comprises an array group.

The format process of a frame creates a logical disk presentation for the host computer. A RAID5 or RAID1 four-disk array group sub-divides itself using an open emulation type. The following table presents the supported emulation types for the most recent XP256 firmware release:

Emulation	Physical Size
Open-3	2.80 GB
Open-8	7.03 GB
Open-9	7.04 GB
Open-E	14.3 GB

HP typically recommends using the disk size presented thru the emulation type on most UNIX based systems. Each emulated type logical devices appears as an individual physical disk to each connected host. Volume management applications, i.e. LVM or VxVM, allow creation of host based meta volumes. Performance testing indicates host based volume management provides higher throughput than frame based volumes. Choosing a single emulation type for the entire frame decreases management of data location and increases data flexibility.

System Bus and Cache Performance and Availability

The XP256 system bus and cache allow high-speed access to data by providing high bandwidth and algorithmic data retrieval. Under typical circumstances this allows server I/O request to be handled at memory access speeds. The XP256 makes use of four system buses including dual-active data buses supporting 252 MB/sec and dual-active command buses. The command buses support data traffic in the event of data bus saturation. The total throughput of the system buses is 744 MB/sec. Every bus on the XP256 is connected to the system cache (maximum of 16GBs). The cache is divided between read and write operations. All write cache is mirrored to support data availability in the event of power loss to the XP256. The XP256 utilizes shared memory (maximum of 512 MBs) for all command tables.

Client Host Interfaces

The XP256 provides connectivity options including: Fiber Channel, FWD SCSI, Ultra SCSI, and ESCON connectivity. The XP256 uses Client Host Interface Processors (CHIPs) to connect the storage system to the server. Each CHIP provides redundancy by utilizing an individual processor per port with the capability to fail-over to a secondary processor in the event of a processor failure. The following table lists the port configuration and performance.

Interconnect	Performance	Ports
Fibre Channel	1024 Mb/sec	4
Ultra SCSI	40 MB/sec	8
HVD SCSI	20 MB/sec	8
ESCON	17 MB/sec	4/8

A Systems Administrator's view of the XP256

The key to successful Storage Management is simplification and flexibility. Now it makes sense why we must view each disk in the array with identical importance. With this view, we can allocate disks for a database, NFS export, or temporary workspace concerning ourselves only with avoiding conflict within a volume group. One of the tools that we can use to visualize the XP256 internals is the following map:

CL1-F / CL2-F		CL1-A / CL2-A		CL1-F / CL2-F				CL1-A / CL2-A		CL1-B / CL2-B		CL1-E / CL2-E	
LUN	3-2	LUN	3-2	LUN	3-1			LUN	2-1	LUN	2-2	LUN	2-3
23 (4.3)	3:1E	23 (4.3)	3:0F	01 (0.1)	3:00			01 (0.1)	1:00	01 (0.1)	1:0F	01 (0.1)	1:1E
24 (4.4)	3:1F	24 (4.4)	3:10	02 (0.2)	3:01			02 (0.2)	1:01	02 (0.2)	1:10	02 (0.2)	1:1F
25 (4.5)	3:20	25 (4.5)	3:11	03 (0.3)	3:02			03 (0.3)	1:02	03 (0.3)	1:11	03 (0.3)	1:20
26 (4.6)	3:21	26 (4.6)	3:12	04 (0.4)	3:03			04 (0.4)	1:03	04 (0.4)	1:12	04 (0.4)	1:21
27 (4.7)	3:22	27 (4.7)	3:13	05 (0.5)	3:04	A	A	05 (0.5)	1:04	05 (0.5)	1:13	05 (0.5)	1:22
29 (5.1)	3:23	29 (5.1)	3:14	06 (0.6)	3:05	C	C	06 (0.6)	1:05	06 (0.6)	1:14	06 (0.6)	1:23
2A (5.2)	3:24	2A (5.2)	3:15	07 (0.7)	3:06	P	P	07 (0.7)	1:06	07 (0.7)	1:15	07 (0.7)	1:24
2B (5.3)	3:25	2B (5.3)	3:16	09 (1.1)	3:07			09 (1.1)	1:07	09 (1.1)	1:16	09 (1.1)	1:25
2C (5.4)	3:26	2C (5.4)	3:17	0A (1.2)	3:08	3	1	0A (1.2)	1:08	0A (1.2)	1:17	0A (1.2)	1:26
2D (5.5)	3:27	2D (5.5)	3:18	0B (1.3)	3:09			0B (1.3)	1:09	0B (1.3)	1:18	0B (1.3)	1:27
2E (5.6)	3:28	2E (5.6)	3:19	0C (1.4)	3:0A			0C (1.4)	1:0A	0C (1.4)	1:19	0C (1.4)	1:28
2F (5.7)	3:29	2F (5.7)	3:1A	0D (1.5)	3:0B			0D (1.5)	1:0B	0D (1.5)	1:1A	0D (1.5)	1:29
31 (6.1)	3:2A	31 (6.1)	3:1B	0E (1.6)	3:0C			0E (1.6)	1:0C	0E (1.6)	1:1B	0E (1.6)	1:2A
32 (6.2)	3:2B	32 (6.2)	3:1C	0F (1.7)	3:0D			0F (1.7)	1:0D	0F (1.7)	1:1C	0F (1.7)	1:2B
33 (6.3)	3:2C	33 (6.3)	3:1D	11 (2.1)	3:0E			11 (2.1)	1:0E	11 (2.1)	1:1D	11 (2.1)	1:2C

CL1-E / CL2-E		CL1-B / CL2-B		CL1-F / CL2-F				CL1-A / CL2-A		CL1-B / CL2-B		CL1-E / CL2-E	
LUN	2-3	LUN	2-2	LUN	2-1			LUN	1-1	LUN	1-2	LUN	1-3
23 (4.3)	2:1E	23 (4.3)	2:0F	12 (2.2)	2:00			12 (2.2)	0:00	12 (2.2)	0:0F	12 (2.2)	0:1E
24 (4.4)	2:1F	24 (4.4)	2:10	13 (2.3)	2:01			13 (2.3)	0:01	13 (2.3)	0:10	13 (2.3)	0:1F
25 (4.5)	2:20	25 (4.5)	2:11	14 (2.4)	2:02			14 (2.4)	0:02	14 (2.4)	0:11	14 (2.4)	0:20
26 (4.6)	2:21	26 (4.6)	2:12	15 (2.5)	2:03			15 (2.5)	0:03	15 (2.5)	0:12	15 (2.5)	0:21
27 (4.7)	2:22	27 (4.7)	2:13	16 (2.6)	2:04	A	A	16 (2.6)	0:04	16 (2.6)	0:13	16 (2.6)	0:22
29 (5.1)	2:23	29 (5.1)	2:14	17 (2.7)	2:05	C	C	17 (2.7)	0:05	17 (2.7)	0:14	17 (2.7)	0:23
2A (5.2)	2:24	2A (5.2)	2:15	19 (3.1)	2:06	P	P	19 (3.1)	0:06	19 (3.1)	0:15	19 (3.1)	0:24
2B (5.3)	2:25	2B (5.3)	2:16	1A (3.2)	2:07			1A (3.2)	0:07	1A (3.2)	0:16	1A (3.2)	0:25
2C (5.4)	2:26	2C (5.4)	2:17	1B (3.3)	2:08	2	0	1B (3.3)	0:08	1B (3.3)	0:17	1B (3.3)	0:26
2D (5.5)	2:27	2D (5.5)	2:18	1C (3.4)	2:09			1C (3.4)	0:09	1C (3.4)	0:18	1C (3.4)	0:27
2E (5.6)	2:28	2E (5.6)	2:19	1D (3.5)	2:0A			1D (3.5)	0:0A	1D (3.5)	0:19	1D (3.5)	0:28
2F (5.7)	2:29	2F (5.7)	2:1A	1E (3.6)	2:0B			1E (3.6)	0:0B	1E (3.6)	0:1A	1E (3.6)	0:29
31 (6.1)	2:2A	31 (6.1)	2:1B	1F (3.7)	2:0C			1F (3.7)	0:0C	1F (3.7)	0:1B	1F (3.7)	0:2A
32 (6.2)	2:2B	32 (6.2)	2:1C	21 (4.1)	2:0D			21 (4.1)	0:0D	21 (4.1)	0:1C	21 (4.1)	0:2B
33 (6.3)	2:2C	33 (6.3)	2:1D	22 (4.2)	2:0E			22 (4.2)	0:0E	22 (4.2)	0:1D	22 (4.2)	0:2C

The map provides a simple means of determining disk configuration and data layout for the storage environment. Historically, large disk frames for storage consolidation focus on allocating disk space to ports directly attached to a host computer. With the introduction of SAN based storage connectivity, the dynamics of storage allocation changed from mapping LUNs to storage ports to determining LUN access for each host.

The initial task in disk layout uses an open emulation type to divide all array groups into a set of logical disk devices. The XP256 utilizes the CU:LDEV nomenclature to address each logical disk device in the frame. The XP256 supports 256 LDEVs and 4 CUs, providing a combined 1024 potential unique devices. The CU:LDEV assignment are manually defined during the formatting process, and the typical practice assigns a unique CU to each ACP and serially increments the LDEVs starting with the initial array group attached to the ACP.

Once the format of the CU:LDEV completes, two paths configured for every CU:LDEV combination supports no single points of failure. A LUN value from 0x00-0xFF for each port uniquely identifies every logical drive. Every LDEV in an array group utilize the same fibre channel port pair. When migrating between array groups a round robin pattern equitably distributes LUNs between all ports. Multi-path software, i.e. LVM PVLinks, provides path fail-over support in the event of a host channel malfunction or cable loss.

The disk frame mapping provides an ability for all hosts connected to the SAN to uniquely address every logical disk within the frame. This allows the system administrator to quickly add logical disk to a system or modify any system's logical disk for clustering or data migration. This flexibility provides granular adjustments for many activities impossible to complete without major disk infrastructure reconstruction.

	CL1-A / CL2-A		CL1-B / CL2-B		CL1-E / CL2-E	
	LUN	1-1	LUN	1-2	LUN	1-3
ACP 0	12 (2.2)	0:00	12 (2.2)	0:0F	12 (2.2)	0:1E
	13 (2.3)	0:01	13 (2.3)	0:10	13 (2.3)	0:1F
	14 (2.4)	0:02	14 (2.4)	0:11	14 (2.4)	0:20
	15 (2.5)	0:03	15 (2.5)	0:12	15 (2.5)	0:21
	16 (2.6)	0:04	16 (2.6)	0:13	16 (2.6)	0:22
	17 (2.7)	0:05	17 (2.7)	0:14	17 (2.7)	0:23
	19 (3.1)	0:06	19 (3.1)	0:15	19 (3.1)	0:24
	1A (3.2)	0:07	1A (3.2)	0:16	1A (3.2)	0:25
	1B (3.3)	0:08	1B (3.3)	0:17	1B (3.3)	0:26
	1C (3.4)	0:09	1C (3.4)	0:18	1C (3.4)	0:27
	1D (3.5)	0:0A	1D (3.5)	0:19	1D (3.5)	0:28
	1E (3.6)	0:0B	1E (3.6)	0:1A	1E (3.6)	0:29
	1F (3.7)	0:0C	1F (3.7)	0:1B	1F (3.7)	0:2A
	21 (4.1)	0:0D	21 (4.1)	0:1C	21 (4.1)	0:2B
	22 (4.2)	0:0E	22 (4.2)	0:1D	22 (4.2)	0:2C

The following example depicts a single ACP with 3 36GB Array Groups configured with RAID5 Open-8. ACP0 receives a CU values of 0, and the LDEV count begins in array group 1-1 at 0x00 and ends in array group 1-3 at 0x2C. Each array group supports 15 LDEVs and hexidecimal notation provides addressing for each LDEV. The example assigns ports CL1-A/CL2-A to all LDEVs in the 1-1 array group. The 1-2 array group receives ports CL1-B/CL2-B, and the round robin pattern continues as you cycle thru the remaining array groups. The XP256 expects a LUN assignment between 0x00 and 0x8F, but all systems represent this as a target id and LUN combination. For example: CU:LDEV 0:14 receives LUN 17 from an XP256, but a combination of target id 2 LUN 4 from a host perspective.

From the host, the inquiry256 or xpinfo tools are instrumental in correlating an array device with a host device. They are the tools that help us make our allocation plan into a reality on the host, and are available from HP in shar format. Execution of the inquiry256 command against a character device provides the following sample data to standard output:

```
% inquiry256 /dev/rdsk/c13t0d1
                /dev/rdsk/c13t0d1 : CL1A     0    1   1:00  00036058

% inquiry256 /dev/rdsk/c22t0d1
                /dev/rdsk/c22t0d1 : CL2A     0    1   1:00  00036058
```

Field 1 is the Host device file for the drive
Field 2 is the FCAL adapter port on the XP256 through which the drive is accessed.
Field 3 is the SCSI Target assigned by the XP256 to the drive
Field 4 is the SCSI LUN assigned by the XP256 to the drive.
Field 5 is the CU:Ldev for the drive. CU corresponds to the ACP pair number. Ldev is the hexadecimal number sequence for the logical device.
Field 6 is the serial number of the array.

Note that both the primary and alternate paths to a physical volume will report identical information for fields 3 – 6.

The inquiry256.ksh script causes all devices to be queried and reported to standard output. By using the output from this program and our allocation map, we can now positively identify a device on the host with a LUN on the array. Using the XP256 allocation map and host-based inquiry tools, we are able to visualize a SAN allocation strategy, and build and operate it from the host. We can now take advantage of the features of the storage array to allocate disks within volume groups for maximum performance and availability.

Allocating Disks for Maximum Availability and Performance

Taking our XP256 map example, we can see that this array has four ACP pairs, each with 4 array groups divided into Open-8 (RAID5) logical devices. Suppose we have a requirement for a 72 GB volume group for a database instance, and an 18 GB requirement for an NFS volume group. We can very easily allocate disks for these groups by following a couple of simple rules:

- Allocate disks evenly between ACP pairs for a volume group.
- Allocate evenly between Open-8 array groups within an ACP pair.
- Plan for growth.

For our example, we could meet the requirements and the first two allocation rules for the database volume group by allocating a device from ACP 0, followed by 1, 2, and 3, then repeating two additional times. For growth, we could set aside an additional logical device from each ACP pair. For the NFS volume group, we follow the same strategy, but need only one device from each of three ACP Pairs (7 X 3 = 21 GB). We'll also allocate the device from ACP pair 3 for growth and even striping. By assuming equal performance from each logical device within an Open-8 array, we can simplify our map, and even make it readable and predictable. If we start at Ldev 0:01 on our sample XP256 map, the allocation would proceed as follows:

Volume group	ACP Pair	Ldev
vgDB	0	0:01
	1	1:01
	2	2:01
	3	3:01
	0	0:10
	1	1:10
	2	2:10
	3	3:10
	0	0:1F
	1	1:1F
	2	2:1F
	3	3:1F
vgDB (reserved)	0	0:2E
vgDB (reserved)	1	1:2E
vgDB (reserved)	2	2:2E
vgDB (reserved)	3	3:2E

Volume group	ACP Pair	Ldev
vgNFS	0	0:03
	1	1:03
	2	2:03
	3	3:03

Total: 4 X 7 GB volumes allocated = 28 GB

Total: 12 X 7GB volumes allocated = 84GB
4 X 7 GB volumes reserved = 28 GB

For the two examples, we have allocated more disk space than was required. This was done to insure array I/O balance and volume group growth capability. These device selections are plotted out on the XP256 map below. Notice how, for both volume groups, the two allocation rules were applied. For vgNFS, one device from each of the four ACP

pairs was allocated. When a second requirement for an NFS volume group comes in, we could easily start at 0:12, and allocate one from each ACP pair, duplicating the I/O of the first volume group. With larger volume groups, such as vgDB, more array groups are involved.

When creating volume groups on the XP256, it makes sense to make use of a little-known LVM feature called the "physical volume group". This feature allows us to organize subsets of the volume group together, and greatly simplifies logical volume creation.

Creating the Volume Group with Physical Volume Groups and Links

Referring back to our example map below, we could organize the first four (remember how we allocated them – 0:01, 1:01, 2:01, 3:01) devices into a physical volume group called "vgDBgrp1". The next set of 4 disks (0:10, 1:10, 2:10, 3:10) could be organized into a group called "vgDBgrp2", and the last set into "vgDBgrp3". We could then organize the vgNFS group into a single group called "vgNFSgrp1. A clever technique in mirroring is to allocate the volumes from one array into physical volume groups appended with "A", and the other array with "B"; e.g., "vgDBgrp1A on one array and vgDBgrp1B on the other.

CL1-F / CL2-F		CL1-A / CL2-A		CL1-F / CL2-F				CL1-A / CL2-A		CL1-B / CL2-B		CL1-E / CL2-E	
LUN	3-2	LUN	3-2	LUN	3-1			LUN	2-1	LUN	2-2	LUN	2-3
23 (4.3)	3:1E	23 (4.3)	3:0F	01 (0.1)	3:00			01 (0.1)	1:00	01 (0.1)	1:0F	01 (0.1)	1:1E
24 (4.4)	3:1F	24 (4.4)	3:10	02 (0.2)	3:01			02 (0.2)	1:01	02 (0.2)	1:10	02 (0.2)	1:1F
25 (4.5)	3:20	25 (4.5)	3:11	03 (0.3)	3:02			03 (0.3)	1:02	03 (0.3)	1:11	03 (0.3)	1:20
26 (4.6)	3:21	26 (4.6)	3:12	04 (0.4)	3:03			04 (0.4)	1:03	04 (0.4)	1:12	04 (0.4)	1:21
27 (4.7)	3:22	27 (4.7)	3:13	05 (0.5)	3:04	A	A	05 (0.5)	1:04	05 (0.5)	1:13	05 (0.5)	1:22
29 (5.1)	3:23	29 (5.1)	3:14	06 (0.6)	3:05	C	C	06 (0.6)	1:05	06 (0.6)	1:14	06 (0.6)	1:23
2A (5.2)	3:24	2A (5.2)	3:15	07 (0.7)	3:06	P	P	07 (0.7)	1:06	07 (0.7)	1:15	07 (0.7)	1:24
2B (5.3)	3:25	2B (5.3)	3:16	09 (1.1)	3:07			09 (1.1)	1:07	09 (1.1)	1:16	09 (1.1)	1:25
2C (5.4)	3:26	2C (5.4)	3:17	0A (1.2)	3:08	3	1	0A (1.2)	1:08	0A (1.2)	1:17	0A (1.2)	1:26
2D (5.5)	3:27	2D (5.5)	3:18	0B (1.3)	3:09			0B (1.3)	1:09	0B (1.3)	1:18	0B (1.3)	1:27
2E (5.6)	3:28	2E (5.6)	3:19	0C (1.4)	3:0A			0C (1.4)	1:0A	0C (1.4)	1:19	0C (1.4)	1:28
2F (5.7)	3:29	2F (5.7)	3:1A	0D (1.5)	3:0B			0D (1.5)	1:0B	0D (1.5)	1:1A	0D (1.5)	1:29
31 (6.1)	3:2A	31 (6.1)	3:1B	0E (1.6)	3:0C			0E (1.6)	1:0C	0E (1.6)	1:1B	0E (1.6)	1:2A
32 (6.2)	3:2B	32 (6.2)	3:1C	0F (1.7)	3:0D			0F (1.7)	1:0D	0F (1.7)	1:1C	0F (1.7)	1:2B
33 (6.3)	3:2C	33 (6.3)	3:1D	11 (2.1)	3:0E			11 (2.1)	1:0E	11 (2.1)	1:1D	11 (2.1)	1:2C

CL1-E / CL2-E		CL1-B / CL2-B		CL1-F / CL2-F				CL1-A / CL2-A		CL1-B / CL2-B		CL1-E / CL2-E	
LUN	2-3	LUN	2-2	LUN	2-1			LUN	1-1	LUN	1-2	LUN	1-3
23 (4.3)	2:1E	23 (4.3)	2:0F	12 (2.2)	2:00			12 (2.2)	0:00	12 (2.2)	0:0F	12 (2.2)	0:1E
24 (4.4)	2:1F	24 (4.4)	2:10	13 (2.3)	2:01			13 (2.3)	0:01	13 (2.3)	0:10	13 (2.3)	0:1F
25 (4.5)	2:20	25 (4.5)	2:11	14 (2.4)	2:02			14 (2.4)	0:02	14 (2.4)	0:11	14 (2.4)	0:20
26 (4.6)	2:21	26 (4.6)	2:12	15 (2.5)	2:03			15 (2.5)	0:03	15 (2.5)	0:12	15 (2.5)	0:21
27 (4.7)	2:22	27 (4.7)	2:13	16 (2.6)	2:04	A	A	16 (2.6)	0:04	16 (2.6)	0:13	16 (2.6)	0:22
29 (5.1)	2:23	29 (5.1)	2:14	17 (2.7)	2:05	C	C	17 (2.7)	0:05	17 (2.7)	0:14	17 (2.7)	0:23
2A (5.2)	2:24	2A (5.2)	2:15	19 (3.1)	2:06	P	P	19 (3.1)	0:06	19 (3.1)	0:15	19 (3.1)	0:24
2B (5.3)	2:25	2B (5.3)	2:16	1A (3.2)	2:07			1A (3.2)	0:07	1A (3.2)	0:16	1A (3.2)	0:25

2C (5.4)	2:26	2C (5.4)	2:17	1B (3.3)	2:08	**2**	**0**	1B (3.3)	0:08	1B (3.3)	0:17	1B (3.3)	0:26
2D (5.5)	2:27	2D (5.5)	2:18	1C (3.4)	2:09			1C (3.4)	0:09	1C (3.4)	0:18	1C (3.4)	0:27
2E (5.6)	2:28	2E (5.6)	2:19	1D (3.5)	2:0A			1D (3.5)	0:0A	1D (3.5)	0:19	1D (3.5)	0:28
2F (5.7)	2:29	2F (5.7)	2:1A	1E (3.6)	2:0B			1E (3.6)	0:0B	1E (3.6)	0:1A	1E (3.6)	0:29
31 (6.1)	2:2A	31 (6.1)	2:1B	1F (3.7)	2:0C			1F (3.7)	0:0C	1F (3.7)	0:1B	1F (3.7)	0:2A
32 (6.2)	2:2B	32 (6.2)	2:1C	21 (4.1)	2:0D			21 (4.1)	0:0D	21 (4.1)	0:1C	21 (4.1)	0:2B
33 (6.3)	2:2C	33 (6.3)	2:1D	22 (4.2)	2:0E			22 (4.2)	0:0E	22 (4.2)	0:1D	22 (4.2)	0:2C

Looking at this organization, we can see that each physical volume group will have identical I/O capability. They are each created in a way to maximize the use of every processor, cache, and I/O chain within the array. We assume that no logical device is more critical than another, and focus on the characteristics of the array to maximize performance. By allocating disks in this fashion, the array almost balances itself, and it becomes very predictable where the next set of logical devices for a volume group should be allocated. Now that we have the allocation plan, we can turn to the host to identify and manage these disks.

The inquiry256 command discussed previously is used to positively identify the allocated disks from any host on the SAN. This command will show each disk and its physical volume link. Once we have positively identified each disk and link and determined their host-based device files, we can create our volume groups and logical volumes.

There are several things that should be considered before creating volume groups on a SAN. The following parameters are very important, as they will limit the future growth and the ability to use other LVM techniques if set too low.

Parameter	Default	Recommended
maxvgs	10	64
max_pe	create disk size	4341
max_pv	16	96

maxvgs is a kernel parameter. In a SAN environment, where we building flexibility, and centrally allocating volume group minor numbers, the default is too low. max_pe will limit the maximum size of any disk extended into the volume group, and max_pv will prevent extending the volume group beyond 16 disks (including physical volume links). By setting the recommended values on every SAN-based host and volume group created, management will be hugely simplified, and room within each volume group is maintained for later use.

When creating the volume group, we can extend disks into the physical volume groups by setting the "-g" option on vgcreate, or, we can edit the /etc/lvmpvg file with our changes. When done, we can create the logical volumes for our application.

Creating and Striping the Logical Volume

If we use the standard LVM-way of creating logical volumes, the system will allocate all the extents for the volume from the first available disk in the group. Since we have organized our disks into identically performing physical volume groups, we should take advantage of this, at a minimum, by setting the "strict" option when creating a logical volume. There is, however a very clever technique that can take full advantage of every disk in a physical volume group, and even improve the write performance of the XP256 on a logical volume.

The physical volume group was created to maximize its I/O capability. Each disk is a RAID5 logical device on an array group, each managed by its own ACP pair. If we create a logical volume that is striped across all four disks in the in a physical volume group, we maximize its performance. Further, the impact of high-I/O logical volumes will be spread across four disks in a group, averaged across all the logical volumes in the group. Striping is essentially another way to ensure identical performance, at the logical volume level. The following sequence is an example of how simple it is to create a striped, then mirrored logical volume on our vgNFS volume group:

1. lvcreate –l 1748 –D y –s g /dev/vgNFS vgNFSgrp1A
2. lvextend –m 1 /dev/vgNFS/lvol1 vgNFSgrp1B

The first command creates a striped 7 GB volume (each Open-8 disk is 7GB) on vgNFSgrp1A in vgNFS. The volume is created one extent at a time across each the four disks in the group. The next command mirrors the volume on the same physical volume group on the second array. If we've allocated our physical volume groups identically on the second array, the mirror for each extent will be created on the exact same position on both arrays. If one array is lost for any reason, the performance of the volume group is maintained. Consider what has now been accomplished:

- Volume groups can be identified and imported on any host.
- Each disk on the array has path redundancy through its physical volume link.
- Logical volumes are treated identically, independent of application.
- Array features are fully exploited for every logical volume.
- The RAID5 write penalty is minimized across a RAID0 4-way stripe.
- The minimized write penalty is offset by a RAID1 set across arrays.

By simplifying the environment, standardizing the way in which storage is allocated and managed, and by designing in redundancy and flexibility, the SAN and Storage Arrays pay big dividends to the HA infrastructure. However, as stated previously, a component failure in the storage space can be difficult to detect and correct with minimal impact.

Troubleshooting Techniques.

The XP256 Storage Array, like any other computer equipment, will have component failures in its lifetime. Most component failures are never seen by any host on the SAN, and can be repaired with no issues:

- Disks
- DKU
- Power Supply
- Console

The issues on an array that can be seen by hosts on the SAN are usually more of a nuisance than an outage:

- Data Center power failure
- FCAL Port failure
- FCAL Switch failure

Strange as it may seem, in an HA infrastructure where the arrays are mirrored, the loss of a Data Center by power or environmental failure has the lowest impact of the three. The worst impact seen during such an event is when a volume group is failed over to another host, activated with the quorum requirement. This is easily fixed by setting the "-q n" option in vgchange. When the failed array returns to life, calling vgsync to resynchronize the mirrors is the biggest recovery effort required.

The failure of a FCAL port or switch is not, by itself, a problem. The hosts report that the ports are "powerfailed", and switch to the alternate physical volume link. This poses no issue, except when attempting to modify an existing volume group. When problems arise, the "-f" option on vgreduce can be used to remove the missing physical volume links.

The worst problems in an HA infrastructure occur when a failed FCAL port on the XP256 is repaired. The reason for this is simple. The XP256 uses 2-port FCAL cards. If one port fails, both must be removed in the repair. In and of itself, this is not a problem. The second port failure simply causes the hosts to use alternate physical volume links. The true problem occurs when the replaced card is put back into action. The FCAL card is much like a network adapter – each has its own "World Wide Name" (WWN) address. If a card is replaced, its WWN is changed. This has a significant impact on the host side, as it addresses I/O to the array based on the WWN. The effect is similar to the network condition where two MAC addresses are using the same IP address. The hosts react differently to this condition:

- HPUX 10.20 systems have a tendency to hang at the LVM level. This can be detected by running "vgdisplay".

- HPUX 11.0 systems typically will not hang, but will produce huge volumes of error messages in the syslog file. Disk performance is decreased.

In both cases, the problems can be cleared up by immediately running "ioscan" on every SAN-connected host. This effectively resets the FCAL bus, removing the error condition.

The simplest techniques to detect and resolve Storage Array issues are as follows:

- inquiry256.ksh. Compare the output from a previously known good output. Missing physical volumes or links can be identified in this way.

- Vgdisplay. On an HPUX 10.20 host, this will tend to hang, indicating (usually) a FCAL port problem.

- Ioscan. On all HPUX hosts, ioscan will reset the FCAL bus and clear mismatched WWN entries.

- Insf –e. On all HPUX hosts, this will rebuild any missing device files.

Summary.

Storage Area Networks and Arrays not only provide significant benefit, but are the very centerpiece of the HA environment. Correct allocation and management of the large Storage Array can provide measurable improvement in the performance and availability of the entire infrastructure. The key to building success in the use of large storage arrays and the SAN is to simplify the administrator's view of the environment and architect flexibility everywhere. Then, management of storage services is reduced to simple Logical Volume Management procedures. By understanding the internals of the Storage Array, we can design tools and methods of disk allocation that take advantage of its features. Using clever Logical Volume Management techniques, the RAID level performance penalties can be overcome, and allocation simplified.