NAS in an HP Environment

Tony Shen

Sr. Consultant
SCSA, SCNA, OCPDBA

July 22, 2002

UNIX Services

Shell Information Technology International, Inc

Acknowledgment

This presentation was inspired, reviewed, and edited by James LeMaire, Jr., one of the finest IT professionals at Shell. As a Sr. Consultant and an HP Focal Point, Jim has worked on and is responsible for numerous high profile and mission critical HP system projects at Shell. A couple of years ago, Jim introduced NAS technology into SITI by putting it into a large HP-UX based Oracle database application landscape as the storage solution. I was fortunate enough to work with Jim as part of the project team. The implementation altered my view toward NAS for good. Like many other people in the IT field, I didn't believe NAS filers could be fast and robust enough to run heavy-duty databases until I see it with my own eyes. That project ignited my interest in NAS. With Jim's help, I looked into NAS much more deeply than a casual NAS implementer would do. The time spent in that regard resulted in, among other things, this presentation. I hope this presentation will help you understand more about NAS and what it can do for your IT. This paper, of course, wouldn't be possible without Jim, without the encouragement and support that I received from Natasha Pitanga, Team Manager of UNIX Primary Support, and Williams Schwartz, Business Manager of UNIX Services as well as the entire UNIX team at Shell. Thanks to Jim. Thanks to my UNIX team!

Contents

Introduction

Storage technology has been undergoing many changes in recent years. One of the most noticeable developments over the last five years is the remarkable growth of networked storage (NS), including both Network Attached Storage (NAS) and Storage Area Network (SAN). According to Gartner and other IT research firms, NS market grew from $580 million in 1997 to $7 billion in 2001. The annual growth rate of the market over the years was as high as over 200%. During that period, NAS growth rate outpaced that of SAN. All major hardware vendors such as IBM, HP and Compaq, Dell, Network Appliances, EMC, and Sun have jumped on the NAS bandwagon and are offering various NAS products today. Among those vendors, Network Appliances is believed to lead in NAS unit shipments and the storage giant EMC claims to have captured the largest revenue share in NAS market in 2001.

Two forces drive the dynamic growth of NAS. One is the rapidly growing demand for data storage raised by Internet centric new business models adopted by many companies. Two is what NAS has to offer in meeting that ever increasing demand for storage, namely, scalability, adaptability, centralization of data storage management, and above all, the relatively low total cost of ownership (TCO) vs. traditional solutions.

This paper discusses how NAS can be used to benefit an HP-UX based landscape. A landscape, in this discussion, is defined as a group of servers that support a particular business mission in an enterprise environment. Furthermore, the landscape is a database-centric server group that runs Relational Database Management System (RDBMS) as its main application.

Methodology

The discussion focuses on topology design involving NAS.

The objective is to answer the following questions concerning NAS in a HP-UX environment:

- What is NAS?
- What needs to be done to support NAS?
- What needs to be done to optimize NAS performance?
- What needs to be done to optimize Oracle performance when using NAS?
- How to build a High Availability (HA) NAS using HP technologies?
- How can Oracle 9i Real Application Clusters (RAC) enhance a HP landscape with NAS?

In order to keep the discussion focused, we will limit ourselves to:

- Preparing HP servers
- NFS and NFS tuning
- Oracle and Oracle file system layout considerations
- Service Guard for Linux and Service Guard NFS for Linux from HP
- Oracle 9i Real Application Clusters

In the discussion, examples are given in order to illustrate the ideas when necessary. The examples are presented as feasible practices rather than real implementation cases. Nonetheless, what is put forth in this paper is completely based on published technical information, white papers and research reports from major vendors and other well established sources on NAS. Extensive research was conducted on various topics related to NAS. References to relevant topics and certain amount of technical details are provided in the appendixes at the end of this paper so as to facilitate reader’s further investigation and verification when they are needed.

What is NAS?

NAS is a way of sharing storage resources at the file level over the network.

NAS provides file sharing over the network using NFS and CIFS protocols. Both protocols run on top of TCP/IP protocol stack. NAS, therefore, is a network application that fits into the top layer, or the Application Layer, of the TCP/IP Five-Layer Model. TCP/IP's dominance in computer networks makes NAS an attractive alternative to other types of networked storage sharing solutions. At the same time, TCP/IP's heavy overhead limits NAS to compete in performance against certain networked storage technologies such as SAN.

From hardware perspective, a NAS device is no more than a server appliance optimized for file sharing. A NAS device typically consists of two parts: a head and a large storage subsystem. The head is often an Intel Architecture (IA) based computer running a kernel that is derived either from Linux and the like or from Windows NT/2000. IA gives NAS a considerable cost advantage over competing technologies using proprietary hardware because NAS devices can be easily built with inexpensive Intel processors and standard components.

The storage subsystem of a NAS device is equally conventional. It uses the same types of disk subsystem or disk arrays that direct attached storage (DAS) architecture uses. Accordingly, interconnects between the head and the storage are rather traditional. NAS storage interconnects fall into three categories:

- ATA/IDE for small NAS
- SCSI for mid-sized NAS
- FC-AL for enterprise-class NAS

NAS is primarily used in Ethernet networks, although the NAS application boundary is starting to blur with NAS increasingly converging with SAN and using new network technologies that may not always be Ethernet.

As an Ethernet-based, TCP/IP device, a NAS system is often equipped with one or more network interfaces that conform to the following network media standards:

- IEEE 802.3z (1000BASE-X, 1000BASE-SX – Gigabit Ethernet over single-mode or multimode fiber optical)
- IEEE 802.3ab (1000BASE-T – Gigabit Ethernet over Cat-5 UTP)
- IEEE 802.3u (100BASE-T – 100MB Ethernet over twisted pair copper)
- IEEE 802.3ae (10GBASE SR/SW, 10GBASE LR/LW, 10GBASE ER/EW, 10GBASE LX4 over fiber optical)

Among the standards listed above, IEEE 802.3z and IEEE 802.3ab are the most commonly used ones by NAS. IEEE 802.3u is the minimum requirement by NAS. IEEE 802.3ae, while still in its nascent stage, represents an emerging technology that NAS can use in order to be able to connect over high speed links such as Synchronous Optical NETwork (SONET) to wide area networks (WAN).

NAS is bilingual. It supports both NFS for UNIX file sharing and CIFS for Windows file sharing. Being able to allow concurrent access by both UNIX clients and Windows clients to commonly shared file systems, NAS sits well in a mixed environment.

Finally Table 1 below gives you a glimpse of state of the art in NAS going enterprise.

Table 1 – **Representative high-end NAS appliances**

CHARACTERISTIC	AUSPEX NS3000	COMPAQ e7000	EMC CELERRA	IBM 300G	NETAPP F880	PROCOM 3000	VERITAS SERVPOINT
Capacity supported (maximum)	12TB	64TB	28TB	22TB	9TB	17TB	Depends on Sun server
Cache memory (max per controller)	3GB	4GB	3GB	2GB	3GB	2GB	Depends on Sun server
Connectivity types	Gigabit, 10/100 Ethernet, ATM	Gigabit, 10/100 Ethernet	Gigabit, 10/100 Ethernet, ATM, FDDI	Gigabit, 10/100 Ethernet	Gigabit, 10/100 Ethernet, ATM	Gigabit, 10/100 Ethernet	Depends on Sun server
Protocols supported	NFS, CIFS, FTP	NFS, CIFS, NCP, AppleTalk	NFS, CIFS, FTP	NFS, CIFS, HTTP, AppleTalk, NCP	NFS, CIFS	NFS, CIFS	NFS, CIFS
Failover (cluster models have 2x capacity, cache, etc.)	Yes - NS3000XA model	Yes - basic	Yes - basic	Yes - Model G26	Yes - F880C model	Yes - 3600C model	Clustered via Sun servers
File sharing	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Locking type implemented	NFS style	CIFS style	NFS or CIFS - user's choice	CIFS style	NFS and CIFS styles	NFS style	NFS style

FEATURES							
PIT copy	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Remote copy	Yes	Host-based	Yes	No	Yes	Yes	Yes

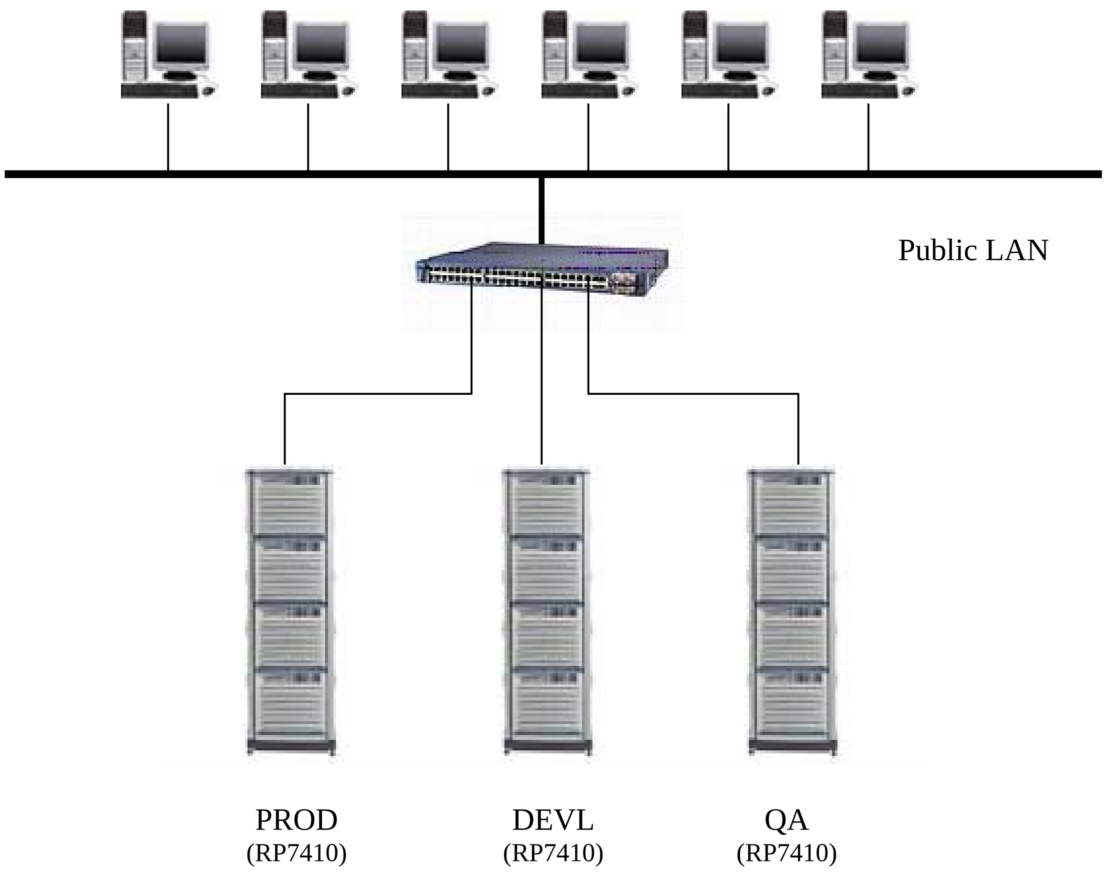
NEW TECHNOLOGY							
DAFS	No	No	No	No	No	No	No
TCP/IP accelerator	Not standard	Not standard	Not standard	Not standard	Not standard	Not standard	Not standard
Metadata server option	No	No	Yes (with HighRoad)	Yes (with HighRoad)	No	No	No
NAS	Yes	Yes	Yes	Yes	No	No	No
Gateway option	Yes	Yes	Yes	Yes	Yes	No	No
LDAP standard	No	No	Yes	No	No	Yes	No
Other	N/A	Virtual software for file system expansion	Up to 14 NAS controllers	N/A	Aggreg. software with Data Fabric Manager	Allows block access to attached disks	N/A

Source: NAS – More than just an appliance by Randy Kerns. Storage, Vol. 1 No. 5 July 2002

DAS Landscape

In order to have a simple start for our discussion, let's have an HP-UX landscape without NAS as depicted in Figure 1. Each server in the landscape has its own storage subsystem directly attached to itself. Therefore, it is a DAS landscape. Possible choices for DAS storage subsystems can be found in Appendix A.

Figure 1 - HP-UX DAS Landscape



In the landscape, three HP rp7410 servers are used with identical configurations. The configuration is shown in Table 2.

Table 2 - HP RP7410 System Profile

Model	**HP RP7410**
CPU	PA8600 * 8
Memory	32GB
OS	HP UX 11i
Internal Disks	SCSI 18GB * 2

Each of the three servers exports part of its file systems and cross-mounts NFS file systems from each other. This way a certain degree of storage resources sharing is achieved.

Each server runs one Oracle database instance. Three Oracle instances, PROD, DEVL, and QA, exist in this landscape.

Oracle clients access the database servers via a 100BASE-T public LAN.

This landscape configuration has the following implications with respect to resource sharing, performance and reliability:

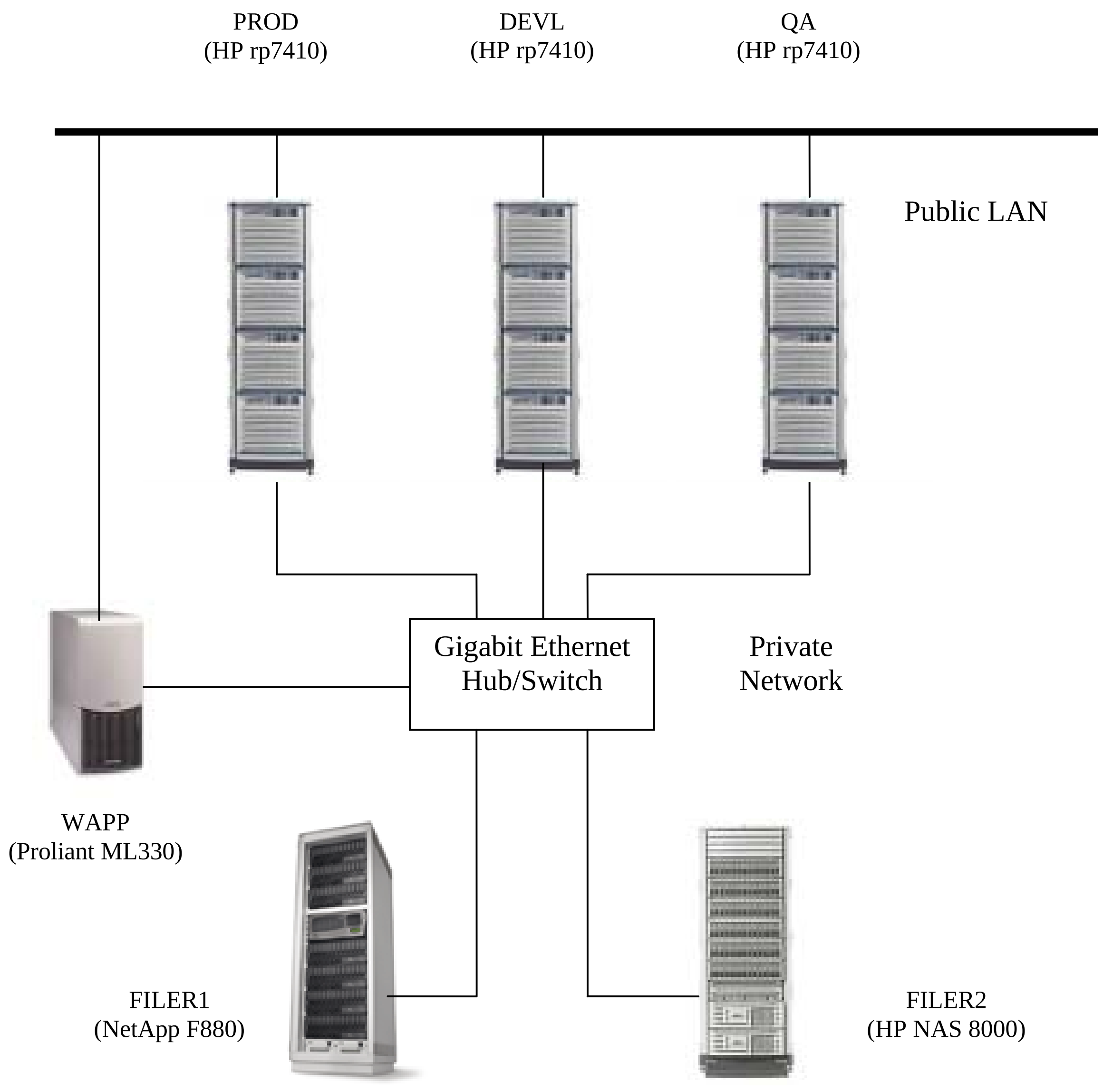
1. Resource sharing and reliability – If an Oracle instance does not use any NFS mounted file systems, it has two points of failure: the server on which it runs and the DAS subsystem that is attached to the server. If an Oracle instance uses NFS mounted file systems from another server, resource sharing increases from 0 to a degree of 2, the points of failure increase from 2 to 4 – the host server, the host DAS, the remote NFS server, the remote DAS. If an Oracle instance uses NFS mounted file systems from the other two servers, storage resource sharing increases from 0 to a degree of 3, the number of points of failure increases from 2 to 6 – the host server, the host DAS, the second server, the second DAS, the third server, and the third DAS. Obviously, resource sharing goes against reliability. It is a dilemma inherent in the design and it is impossible to overcome.
2. Performance - If the Oracle instances share storage resources via NFS, each instance's performance is impacted by the network traffic and disk I/O contention that is bound to occur on both the local DAS and remote DAS subsystems. For example, when QA is accessing a NFS file system on PROD, PROD is busy serving Oracle PROD causing disk I/O to the underlying disks of the NFS file systems, the contention will not only slow down QA but also PROD. To dedicate the underlying disks to QA is not a sensible option because you would be much better off by adding disks to QA's local DAS to satisfy your needs.
3. Each system's storage has to be managed separately, which adds administration overhead. Disk fault tolerance such as RAID can only be implemented within each DAS not across the DAS's.
4. Network becomes a performance bottleneck – User processes that connect to Oracle instances will contend for the limited bandwidth with NFS data traffic. One user process can generate multiple NFS requests and responses from NFS servers at any given point of time. User processes that run extended query and updates can easily saturate the network when such processes multiply during peak time. Without a high-speed network in place, sharing storage resources via NFS is not recommended for disk I/O and file I/O intensive Oracle database application at all.

NAS Landscape

How can NAS be used to address the DAS problems? Let’s change the design by incorporating NAS and examine what differences NAS will bring to the landscape.

Figure 2 shows a revised, NAS based configuration for the landscape.

Figure 2 NAS-based Landscape



In this configuration, DAS is replaced by two NAS filers. One is F880 from Network Appliance; the other is HP NAS 8000 from Hewlett Packard. Both filers and the three rp7410 servers are on one Gigabit Ethernet private network. This private network needs to be an 802.3ab or 802.3z Ethernet or of a higher bandwidth so as to ensure sufficient network throughput between the filers and servers.

The two NAS devices are similar in a number of ways. Table 3 below compares a few features between the two filers.

Table 3 Comparison between NetApp F880 and HP NAS 8000

Features/Specs	NetApp F880	HP NAS 8000
OS Kernel	Free BSD	Linux
Storage topology	FC-AL	FC-AL
Maximum storage capacity	9.0 TB	15.4 TB
Recommended Network Interface	Gigabit Ethernet (803.3ab/803.z)	Gigabit Ethernet (803.3ab/803.z)
OS User Interface	Ontap	Command View
Snapshot feature sets	Standard	Optional (Business Copy)
Native bilingual support for NFS and CIFS	Yes	Yes

Sources: Network Appliances, Hewlett Packard

The three servers NFS mount file systems from the two filers as follows:

On PROD

/export/prod	(local disk file system)
/data/filer1data	(filer1:/vol/data)
/data/filer2data	(filer2:/vol/data)

On DEVL

/export/devl	(local disk file system)
/data/filer1data	(filer1:/vol/data)
/data/filer2data	(filer2:/vol/data)

On QA

/export/qa	(local disk file system)
/data/filer1data	(filer1:/vol/data)
/data/filer2data	(filer2:/vol/data)

The introduction of NAS accords a number of benefits to the landscape.

- The data traffic is separated from user traffic, thus reducing contention on the public network. The filers share their resources via a dedicated high speed network. A certain level of performance is assured.
- The storage resource sharing is the default. All three servers have equal access to the filers. Storage resources can be more centrally managed on the filers.

Separation of disk resources is reduced from the previous three DAS's to two filers, which allows better allocation and utilization of disk space as collective resources. Further consolidation of disk resources on only one filer is always possible.

- Scalability is enhanced. When business grows, it is easier to add disks to the filers than adding disks to individual servers. The added resources can be made available instantly to all three servers without hassle.
- It gives more flexibility to DBA to partition their databases. For example, Oracle PROD database can be partitioned not only across multiple disk controller channels within one filer but also between the two filers to further spread out disk I/O. Similarly, disk I/O can be more easily managed on filers than on individual servers. For instance, if it is so observed that DEVL and QA databases incur much less disk I/O than PROD, DBA can adjust data file locations that involve both DEVL and QA so that they use fewer disk channels. The released disk channels on the filers can then be used for PROD to distribute her work load across more evenly. Such adjustment would be very hard to do, if not impossible, in a DAS landscape.
- It simplifies backup and restore because the data storage is consolidated. Moreover, snapshot capability provided by filers equips DBA with new means to perform online backup and database recovery. Better backup and restore infrastructure weighs in favor of mission-critical databases.
- It enables the landscape to natively support non-UNIX clients such as Windows. Concurrent NFS and CIFS access to the same file systems on one filer is permissible. File locking mechanism is natively supported by the filers.

Server Requirement for NAS Landscape

As NAS brings benefits, it imposes certain requirements on HP-UX servers. A number of server requirements ought to be met when implementing the NAS landscape in discussion.

These requirements are

- OS patches
- Kernel Configuration
- Network Interface Configuration
- NFS Client Configuration

Let's take a look at each requirement in the following sections.

OS Patches

We recommend HP-UX 11i operating system for the servers becauase it has a number of important enhancements that we need over its predecessors.

When using 11i, you need to install the OS patches listed below to make the server ready to work with NAS. All patches are network related patches.

Since HP releases new patches almost every day. The patches listed below may replaced by new ones. Always consult with HP for newest patch releases when you are ready to start preparing your servers.

- Install OS Patch PHNE 25652. It is an NFS problem fix and performance enhancement patch. It supersedes patches PHNE 24910 and PHNE 24035. Unlike PHNE 24910, however, no special enabling procedure is required when applying this patch.
- Install OS Patch PHNE 22962. This patch supersedes PHNE 21217. After applying PHNE 22962, you may also apply the most recent patch PHNE 26250. Unlike PHNE 22962, PHNE is not a cumulative patch. In other words it does not supercede PHNE 22962 entirely. These patches are to fix problems associated with PCI 1000Base-T or HSC/PCI 1000Base-SX LAN Cards, which are used on RP7410 server systems in this example.

PCI 1000Base-T LAN Card supports 802.3ab standard using Category 5 twisted pair cable. HSC/PCI 1000Base-SX LAN Card supports 803.z standard using fiber optical multimode cable.

For more details about how to use the cards, please consult "Using PCI 1000Base-T and HSC/PCI 1000Base-SX (Gigabit Ethernet)" document provided by HP.

- Install Patch 26551 if you run HP UX 11i 64-bit, Service Guard, and Hyper Fabric Cluster on the servers

Kernel Tuning

The kernel parameters listed in Table 4 below are the minimum kernel requirements for Oracle9*i*. If you have previously tuned your kernel parameters to levels equal to or higher than these values, continue to use the higher values. Kernel changes require a system reboot.

In the table, NPROC is the number of user processes. It is determined by one of the Oracle instance initialization parameter PROCESSES.

The PROCESSES initialization parameter determines the maximum number of operating system processes that can be connected to Oracle instance concurrently. The value of this parameter must be 6 or greater (5 for the background processes plus 1 for each user process). For example, if you plan to have 50 concurrent users, set this parameter to at least 55.

Table 4 HP-UX 11i Recommended Kernel Parameters for Oracle 9i

Kernel Parameter	**Setting**	**Purpose**
KSI_ALLOC_MAX	(NPROC * 8)	Defines the system wide limit of queued signal that can be allocated.
MAXDSIZ	1073741824 bytes	Refers to the maximum data segment size for 32-bit systems. Setting this value too low may cause the processes to run out of memory.
MAXDSIZ_64	2147483648 bytes	Refers to the maximum data segment size for 64-bit systems. Setting this value too low may cause the processes to run out of memory.
MAXTSIZ	1073741824 bytes	Equal to MAXDSIZ
MAXTSIZ_64	2147483648 bytes	Equal to MAXTSIZ_64
MAXSSIZ	134217728 bytes	Defines the maximum stack segment size in bytes for 32-bit systems.
MAXSSIZ_64	1073741824	Defines the maximum stack segment size in bytes for 64-bit systems.
MAXSWAPCHUNKS	(available memory)/2	Defines the maximum number of swap chunks where

		SWCHUNK is the swap chunk size (1 KB blocks). SWCHUNK is 2048 by default.
MAXUPRC	(NPROC + 2)	Defines maximum number of user processes.
MSGMAP	(NPROC + 2)	Defines the maximum number of message map entries.
MSGMNI	NPROC	Defines the number of message queue identifiers.
MSGSEG	(NPROC * 4)	Defines the number of segments available for messages.
MSGTQL	NPROC	Defines the number of message headers.
NCALLOUT	(NPROC + 16)	Defines the maximum number of pending timeouts.
NCSIZE	((8 * NPROC + 2048) + VX_NCSIZE)	Defines the Directory Name Lookup Cache (DNLC) space needed for inodes. VX_NCSIZE is by default 1024.
NFILE	(15 * NPROC + 2048)	Defines the maximum number of open files.
NFLOCKS	NPROC	Defines the maximum number of files locks available on the system.
NINODE	(8 * NPROC + 2048)	Defines the maximum number of open inodes.
NKTHREAD	(((NPROC * 7) / 4) + 16)	Defines the maximum number of kernel threads supported by the system.
NPROC	4096	Defines the maximum number of processes.
SEMMAP	((NPROC * 2) + 2)	Defines the maximum number of semaphore map entries.
SEMMNI	(NPROC * 2)	Defines the maximum number of semaphore sets in the entire system.
SEMMNS	(NPROC * 2) * 2	Sets the number of semaphores

		in the system. The default value of SEMMNS is 128, which is, in most cases, too low for Oracle9 *i* software.
SEMMNU	(NPROC - 4)	Defines the number of semaphore undo structures.
SEMVMX	32768	Defines the maximum value of a semaphore.
SHMMAX	Available physical memory	Defines the maximum allowable size of one shared memory segment. The SHMMAX setting should be large enough to hold the entire SGA in one shared memory segment. A low setting can cause creation of multiple shared memory segments which may lead to performance degradation.
SHMMNI	512	Defines the maximum number of shared memory segments in the entire system.
SHMSEG	32	Defines the maximum number of shared memory segments one process can attach.
VPS_CEILING	64	Defines the maximum System-Selected Page Size in kilobytes.
MAX_THREAD_PROC	1024	When using MTS
MAXFILES	2048	When having multiple instances

Network Interface Configuration – Using Jumbo Frames

Each server has two network interfaces. One connects the server to the public LAN. The second one connects between the server and a filer in the private Gigabit Ethernet network. We use an HSC/PCI 1000-SX LAN card for each server's second interface. When configuring this network interface, it is necessary to use Jumbo Frame to boost performance by setting MTU (Maximum Transfer

Unit) from a standard 1500 bytes to 9000 bytes. MTU being set to 9000 bytes is commonly referred to as Jumbo Frames.

The LAN card configuration file is located at /etc/rc.config.d/hpgelanconf. The settings in the file look like these:

```
HP_GELAN_INTERFACE_NAME[0]=lan1
HP_GELAN_STATION_ADDRESS[0]=
HP_GELAN_SPEED[0]=
HP_GELAN_MTU[0]=9000
HP_GELAN_FLOW_CONTROL[0]=1
HP_GELAN_AUTONEG[0]=1
HP_GELAN_SEND_COAL_TICKS[0]=1000
HP_GELAN_RECV_COAL_TICKS[0]=0
HP_GELAN_SEND_MAX_BUFS[0]=16
HP_GELAN_RECV_MAX_BUFS[0]=1
```

Please note that in the configuration, auto-negotiation is turned on. We'll revisit auto-negotiation issue later in our discussion.

You can use 'lanadmin' to configure these settings online. However, unlike using this configuration file, online settings can't survive a system boot.

When using Jumbo Frames, make sure your switch or hub used in the private network connecting between your servers and NAS devices support Jumbo Frames as well. Otherwise you have to use standard frames. If you use direct connect topology, this condition does not apply.

NFS Client Mount Options

Each server is an NFS client to the NAS filers. On each NFS client, proper NFS mount options have to be set in /etc/fstab file. On PROD, DEVL, and QA, the /etc/fstab file needs to contain these entries:

```
filer1:/vol/data - /data/filer1data nfs - -
rw,bg,hard,intr,rsize=32768,wsize=32768,proto=udp,vers=3,suid

filer2:/vol/data - /data/filer2data nfs - -
rw,bg,hard,intr,rsize=32768,wsize=32768,proto=udp,vers=3,suid
```

Please note that each entry is one continuous line rather than two lines that appear to be on this page.

The NFS mount options used in the /etc/fstab entries are:

- rw – allows NFS client read and write access
- bg – retries in background when connecting to NFS server
- hard – uses hard mount instead of soft mount. NFS supports two types of mounts - hard mount and soft mount. If a mount is a hard mount, an NFS request affecting any part of the mounted resource is issued repeatedly until the request is satisfied (for example, the server crashes and comes back up at a later time). When a mount is a soft mount, an NFS request returns an error if it cannot be satisfied (for example, the server is down), then quits. Since hard mount offers the desired client behavior for Oracle, this option is always used.
- intr – allows user interruption
- rsize=32768 – Read buffer size (bytes). The default is 8192. Quadrupling the default read buffer size is meant to improve read performance.
- wsize=32768 – Write buffer size (bytes). The default is 8192. Quadrupling of the default write buffer size is intended to enhance write performance.
- proto=udp – Uses low-overhead UDP rather than high-overhead TCP. This is the default.
- ver=3 – Uses NFS version 3. Both HP UX 11i and NetApp F880 and HP NAS 8000 filers supports NFS Version 3.
- suid – Honors imported SUID and SGID execution permissions as they are set in the remote filesystem. This option is often required by Oracle, particularly for those file systems where Oracle executables reside.

NFS Client Tuning

In this section we will analyze NFS mechanism on the client before we actually tune NFS on each HP rp7410 server.

NFS Daemons

NFS is a true “client/server” application, in order to obtain optimal NFS performance, tuning needs to be done on both the server and the client. In our case, the NFS server is NAS filer(s); the client is HP rp7410 server(s).

Table 5 lists the NFS daemons that run on the NAS filers (NFS Server) and/or on the HP rp7410 servers (NFS Client).

Table 5 – NFS daemons

Daemon	Does it run on NFS Server?	Does it run on NFS Client?
nfsd	Y	N

rpc.mountd	Y	N
biod	N	Y
rpc.lockd	Y	Y
rpc.statd	Y	Y

The tuning, therefore, involves properly configuring the daemons that run on both sides. In this section, we focus on biod daemon configuration since it is the primary NFS daemon that impacts the client (HP rp7410) NFS performance most.

biods are implemented as user-space processes. They spend the majority of their time, however, running in the kernel. Their sole purpose is to try to increase the performance of remote file access by providing read-ahead and write-behind semantics on the NFS client. In most cases they can dramatically improve NFS read and write performance.

In HP-UX 11.0 the default number of biods launched at boot is 4. In 11i, the number is increased by four-fold to 16.

“Read-Ahead and Write-Behind” by biods

When the client needs to read data from an NFS mounted file, the client will first check its local buffer cache to see if the block of data is present in the cache. If it is then the read is satisfied without generating an NFS request. If the data is not present then an NFS READ call will be made to retrieve this data. If no biods are running then the process requesting the data will generate the NFS read call in its own process context.

If biod daemons are present then the client process will send the initial read request in its own context through biods to the server. The client process will then block future read requests to be handled by the biods that it uses. The blocking mechanism is done in conjunction with rpc.lockd and rpc.statd. During the block, the biods employed by the client process send additional sequential reads to the server. Once the biods retrieve the data from the serer, the data is cached in client’s buffer cache and the blocking process is notified that the data is available. Because of the extra reads that the biods sent on behalf of the client process, the cached data includes more than what the client process originally asked for. The extra data is obtained in hope that the client process may need it next time so that the biods don’t have to send another read request to the server but letting the buffer cache to satisfy the client process read request. This process is called “**read-ahead**”.

When the client process needs to write data to an NFS mounted file, it will write the data to the local buffer cache. It then generates a biod if no biods are available to send the data from the buffer cache to the server. When the biods that the client process uses are sending the data through NFS WRITE calls to the server, the client process continues writing more data to the buffer cache. The biods always write to the server after the client process writes its data into the local buffer cache. This process is called "**write behind**". The client process only waits until the biods that it uses finish their NFS writes when

- The local buffer cache is full
- The client process needs to flush, sync, or close the file

The biod "read ahead/write behind" semantics described above clearly indicates that the number of biods has direct bearing on the client NFS performance. When there are not enough biods to populate the local buffer cache for sequential reads or the available biods can't write the data from the buffer cache to the NFS server quick enough for the client process to continue its writes to the local buffer cache, it is beneficial to increase the number of biods.

HP-UX 11i Enhancement for biods

On HP-UX 11.0, setting a high number of biods in order gain NFS performance is not a good idea because the client's read() and write() paths use the global file system semaphore to protect many kernel data structures and NFS I/O operations. According to Dave Olker, Systems Network Solutions Lab of HP, whenever a biod processes a read or write request it must acquire the file system semaphore, which effectively locks out all other file system related operations on the system – not only just to the NFS requests but also to requests for all file systems (i.e. VxFS, HFS, NFS, CDFS, etc.). Therefore, if an 11.0 NFS client runs a large number of biod daemons, the overhead involved with contending for and acquiring the file system semaphore becomes detrimental to NFS and general file system performance.

This file system semaphore contention issue is drastically reduced in 11i, which means that an 11i client could potentially benefit running more biods than an 11.0 client.

Determining the Right Number of biods

One thing that is very interesting to note from Dave Olker's research is that while you can set a high number of biods on HP-UX 11i without worrying much about file system semaphore contention, if the number is not set high enough to serve all requesting client processes, performance will suffer rather than gain. That is because the available biods can serve only a definite number of client process requests, including both read() and write() request, at any given point of time. If the client requests outnumber the biods, they have to wait for their turns. This “queuing” effect will hurt the client NFS performance. Ironically, if there are no biods available at the time, each client request will send NFS request on its own to respective NFS servers without waiting, resulting in a higher throughout.

Adjusting the number of biods may not produce a desired effect on NFS performance when:

1. Client processes perform mainly non-sequential reads - When the client read requests are primarily non-sequential and the odds that next reads will hit any pre-cached data by biods are very low, the “read aheads” by biods become unwanted overhead hurting performance.

2. biods do not play a role in improving NFS performance when read and write requests have to be performed synchronously. Oracle DB Writer (DBWR), Redo Log Writer (LGWR), and Archive Log Writer (ARC0) perform synchronous writes. (We will see how those Oracle background processes work and how do they relate to performance in Oracle Performance Tuning section)

How many biods should the NFS clients (rp7410s) run against the NFS servers (NAS filers) in our landscape?

Recommendation

As a starting point, it is a good idea to set the number of biods equal to the number of PROCESSES that you specified in your instance initialization parameter files (pfiles). Then add 4 to the number allowing for a reserve pool. That number should pretty much cover for the need by all concurrent user processes and server processes. If you have multiple instances, the number, of course, needs to be increased accordingly so as to cover the biod requirement by all instances.

For example, in practice, setting the number of biods to 64 yields pretty good performance for a server running one Oracle instance that supports 50 concurrent users.

Use tools for NFS performance analysis and trouble-shooting to observe the actual performance when Oracle is running and adjust the number of biods accordingly until you reach an optimal point.

The number of biods is configured via the NUM_NFSIOD variable in the /etc/rc.config.d/nfsconf file.

The commonly used tools for NFS performance analysis and trouble-shooting are nfsstat, tusc, and kgmon. Refer to your man page for how to use those commands.

Appendix B provides more resources on NFS tuning issues.

Network Requirement for NAS Landscape

Autonegotiation

If you are using a switch, make sure the switch also provides support for jumbo frames. Many even high-priced switches currently do not. Policy varies from vendor to vendor.

Also note that auto-negotiate is turned on at the server host. The Gigabit Ethernet adapter used in the filer does not allow you to turn off autonegotiation, hence you should not attempt to do so on the host! Autonegotiation settings should always be symmetrical between a filer, a host, and any switches they might be connected to.

HP PCI/HSC GB LAN cards require autonegotiation.

Network Throughput Test

Network Appliance recommends running performance test using ‘dd’ and ‘sysstat’, an Ontap performance statistics utility to gauge the network throughput between the filer and its NFS clients.

On PROD in our landscape, to test write operation with ‘dd’, use this command:

```
dd if=/dev/zero of=/data/filer1data/foo_out bs=32k count=16384
```

To test read operation from PROD:

```
dd if=/data/filer1data/foo_in of=/dev/null bs=32k
```

To observe the filer’s performance with two-second intervals when running write test and read test one after another on PROD, type this command:

```
prod # rsh filer1 sysstat 2
```

To test mixed I/O, run the write and read tests simultaneously:

```
dd of=/data/filer1data/foo_out bs=32k count=16384
dd if=/data/filer1data/foo_in of=/dev/null bs=32k
```

Keep it in mind that when testing mixed I/O the traffic goes over the same mount point on the same volume on the same filer supported by one Gigabit Ethernet interface in the filer.

In the sysstat output, the column 'Net kB/s in' stands for the input from the network into the filer in kilobytes per second. The numbers under this column are for writes from PROD. The column 'Net kB/s out' stands for output from the filer to the network in kilobytes per second. The numbers under this column are for reads from the filer into PROD.

Fluctuation in value under these two columns of more than 50% over 2 iterations can indicate a network problem.

Repeat the above test on DEVL and QA to see if the test results are consistent. If not, it indicates a network problem.

This test procedure can be adapted to run on a NAS device such as HP NAS 8000 by substituting Ontap “sysstat” for NetApp with a corresponding performance analysis utility or set of utilities to gather performance statistics. The utilities that can be used for this purpose are:

top, vmstat, ps

In addition, various system performance monitoring tools for Linux are available on the Internet. One package is ‘sysstat’ available from a Linux performance tools web site at http://linuxperf.nl.linux.org/links.html . The package can be installed as either a set of tar.gz files or as a RPM loadable module that includes sar, mpstat, iostat, and sa tools.

NAS Filer Performance Test

The network throughput is a function of the network, the server hosts, and the clients. Any non-performing component will affect the overall network throughput. Therefore, when you test your NAS network, there are times when you need to look into how the NAS filers are doing. When your network throughput test yields less than satisfactory results and you suspect the NAS filers may contribute more to those tarnished results, you may use “postmark” to investigate the NAS filer’s performance.

Jeffery Katcher wrote “postmark” under contract to Network Appliances. The C program is distributed under “Artistic License” without restrictions on copying, adaptation, and use for test purposes like ours.

The program does the following:

- Offers a command line interface (CLI) to user to run it either interactively or in a batch mode with a command file
- The command file shall contain valid commands that user wants it to execute. If no arguments list is given in the command line as the command file(s), the system will search the file .pmrc at the current directory by default.
- Parses the commands read from the file line by line and terminates when it encounters ‘quit’ command

- Prints “pm>” prompt when no “quit” is found at the end of the command input file or no command file is given.
- Runs a set of file system operations and records the times each operation takes. At the start, the program initializes the counters and generate a random number as the seed. It then allocates buffer space for reads and writes. It fills the allocated space with random data. The file system operations that the program runs include creating files, reading each file into the buffer, writing to each file from the buffer, creating subdirectories under the current directory recursively, deleting files, deleting subdirectories. It runs these operations in a random order instead of sequentially. Time each operation as it goes. At the end, prints a report showing the file system performance in response to these operations on the filer.

The program’s source code can be found at Appendix C for the completeness of this discussion.

Understanding Oracle

As the center piece in our landscape, Oracle performance is our primary concern. Using NAS as the Oracle storage media introduces different issues for DBA and SA to consider than using DAS. Although Oracle performance tuning is a huge topic way beyond our scope, one thing is certain: how do you layout file systems for your Oracle with NAS determines how your Oracle will perform to a quite large extent.

In this section, we will take a three-step approach to examining how we can make NAS fit into Oracle requirement as far as file systems and disk I/O are concerned. First we will need to understand Oracle's architecture and its disk I/O behavior that the architecture dictates. Secondly, we will learn about three file system layout recommendations based on our understanding of Oracle. Last but not least, we will take a look at Oracle Storage Compatibility Program and be aware of what OSCP means to you when you implement NAS.

Oracle Architecture and Oracle Disk I/O Distribution

An Oracle database provides its information management services through an Oracle instance. One Oracle instance manages one database. An Oracle instance is a memory structure called System Global Area (SGA) plus a number of background processes running on a host server. An Oracle instance running on a host machine is often referred to as an Oracle database server. There can be multiple instances managing multiple databases respectively running on a single host machine.

Figure 3 on Page 21 shows Oracle architecture in a diagram that explains Oracle instance and its components.

Oracle is a client/server application. Oracle instance constitutes the server. Applications that use services provided by the instance are Oracle clients. There are two kinds of Oracle clients, local clients and remote clients. Local clients are those who run on the same host machine where the Oracle instance that they connect to is running. Remote clients, on the other hand, are those who run on different machines connecting to the Oracle instance through the network. A client to Oracle instance is a user process.

Before a client can use a database managed by an Oracle instance, the client has to start a session with the instance. The client sends a connection request to his target instance and the instance responds to the request by spawning a server process that authenticates and then allows the client to establish a session with the instance. The server process manages the session, after it is successfully established, until the session terminates.

During a session, the client sends two types of requests to the instance. One is database query (read); the other is database update (write). The instance serves these two types of requests by involving different processes, which cause different disk I/O activities.

Oracle Reads

When the client sends a database query or a read request, the server process responds. It first looks up the library cache, the data dictionary cache and the data buffer in the SGA of the instance to see if the information requested is already in there. If it is, the server process returns the result to the user process without a disk read. If the server process doesn't find the cached results, it will perform a disk read to fetch the information it needs from data files. The fetched results are cached in SGA in anticipation that they may be requested again in future.

There are a number of areas database query performance can be tuned. Those areas are:

1. DBA can adjust shared pool size so as to indirectly change library cache and data dictionary cache sizes to improve performance

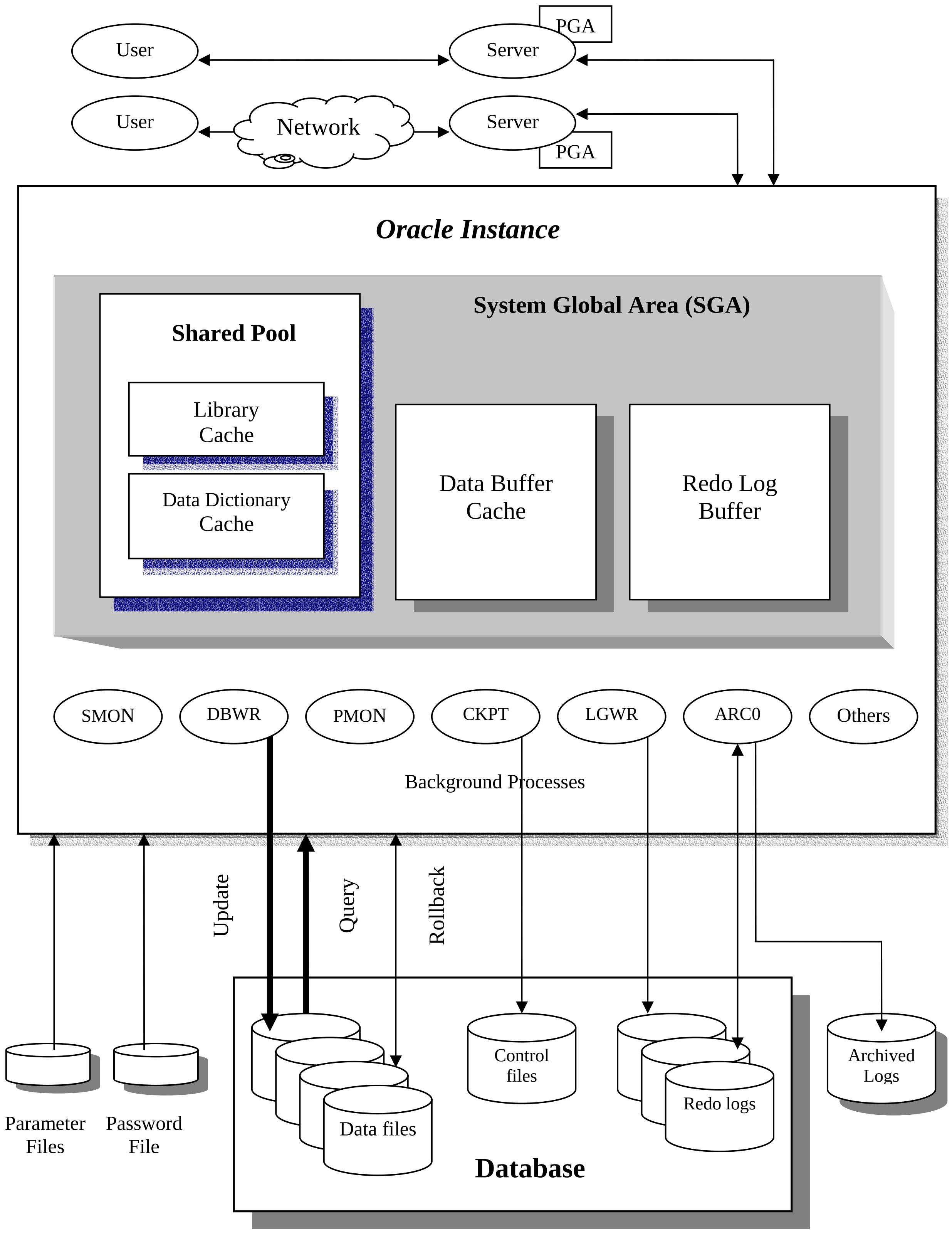
2. DBA can manipulate the data buffer size to optimize performance.

3. DBA can partition the query intensive table(s) across disks and channels to reduce disk read contention.

4. System Administrator can use RAID 0, RAID 1, RAID 0+1, RAID 1+0 that offer better read performance in the storage subsystems.

5. Tune network performance

Figure 3 Oracle Architecture



Oracle Writes

Things are more complicated when the instance serves a database update or a write request. In database terms, DML (Data Manipulation Language) is used to accomplish a database update. DML includes INSERT, DELETE, UPDATE, and DROP statements in PL/SQL. Execution of DML involves not only server process but also Redo Log Writer, Archive Log Writer, Checkpoint, and most importantly, Database Writer processes. Let's inspect the role that each process plays in executing DML and identify performance bottlenecks along the way.

1. Server process – When a client or user process requests to make a change in the database by submitting a DML statement, the server process responds by performing three things: 1) it looks up the data buffer to see if the data that needs to be changed is already in there. If it is not, the server process reads from data files into the data buffer. A disk read is performed. 2) It records the changes it's about to make in the redo log buffer. 3) It makes changes in data blocks and rollback blocks in the data buffer. Data blocks hold the database updates. Rollback blocks keep the before change image for undo or rollback when it is needed. The data blocks in the data buffer that have been changed are called "dirty blocks" because they are not the same as they were in the disk image. Up to this point, all the changes that a server process has made are kept in SGA. Nothing has been written to disk yet. It holds true for every server process when it works with its respective client user process and responds to a client write request. If the instance crashes at this point, all the changes that server processes made in SGA will be lost.

2. When user commits a change, Redo Log Writer (LGWR) writes the changes that the server process recorded in the redo log buffer out to online redo logs on disk. Unlike the server process, when Log Writer writes, it writes to disk. Besides COMMIT event, Log Writer writes out changes in the redo log buffer to online redo log files when any of the following events occurs:

    a. Redo log buffer is one third full;

    b. When there is more than a megabyte of changes recorded in the redo log buffer;

    c. Before DB Writer writes modified blocks, including both data blocks and rollback blocks, to data files. Database Writer waits for Log Writer to complete its writes before it starts its own disk write.

Because the redo is needed for recovery, Log Writer confirms COMMIT only after the redo is successfully written to disk. LGWR writes, therefore, are expensive synchronous disk writes.

3. When archiving is enabled, Archive Log Writer (ARC0) archives online redo log files when Log Writer performs a log switch. When ARC0 performs the archiving, Log Writer waits until archiving finishes. In a busy production database, there are brief moments when DB Writer waits for Log Writer to finish online redo log writes and Log Writer in turn waits for Archive Log Writer to complete redo archive log writes. When waits like these occur, performance suffers.

4. DB Writer writes the dirty blocks out from the data buffer to data files from time to time. Occasionally DB Writer is delayed by Log Writer as we explained previously. Other than that, how often DB Writer writes to disks are determined by the following events, each of which triggers DB Writer to perform an expensive synchronous disk write:

    a. The number of the dirty blocks reaches a threshold value;

    b. A server process requests for free buffers;

    c. Every 3 seconds (a timeout occurs)

    d. A checkpoint occurs.

    Keep it in mind, in NAS environment, when DBWR writes data to disk, the database does not always get updated immediately on the persistent media. The changes can be cached in Non Volatile RAM (NVRAM) implemented in a NAS filer. NVRAM caching is designed to improve disk performance by making DBWR disk writes a little more asynchronous.

5. Checkpoint is a process responsible for keeping the database in sync with the instance data buffer.

As we can see, when an Oracle instance is active, most disk reads are performed by server processes serving user process requests, most writes are done by Redo Log Writer, DB Writer, Archive Log Writer if archiving is enabled, and Checkpoint processes.

Table 6 shows the relationship between disk I/O and Oracle processes

Table 6 - Oracle Disk I/O

Process	**Disk Read**	**Disk Write**	**Disk File(s)**
Server/User Processes	Yes	No	Data Files
Redo Log Writer (LGWR)	No	Yes	Online Redo Logs
Checkpoint	No	Yes	Control Files
DB Writer (DBWR)	No	Yes	Data Files
Archive Redo Log Writer (ARC0)	Yes	Yes	Archive Redo Log Files
Instance Initialization	Yes	No	Parameter Files, Password File (if external password file is implemented)

Care needs to be taken in planning Oracle database file systems on NAS to avoid disk contention between data files, redo log files, and archive redo log files if archiving is to be enabled.

Moreover, the busy tables, i.e., those tables that are read and updated most often such as Sales, Orders, and Inventory, and their associated indexes need to be properly partitioned so as to spread out disk I/O activities across multiple disk devices on multiple channels, or even multiple NAS filers in order to minimize disk I/O contention.

Recommended File System Layout Designs for Oracle

There are three recommendations. Each emphasizes on a different priority. They are:

- The performance orientated – Aimed to maximize performance
- The management orientated – Aimed to simplify maintenance
- The balanced – Talk a middle way

The three approaches share one thing in common – Oracle Home ($ORACLE_HOME), i.e., the Oracle software, is installed on each rp7410 server where each instance runs instead being installed on NAS.

A description for each design follows.

- Performance Oriented - Place the instance initialization files (pfiles), the control files, and online redo log files, process log files, and user log files on each rp7410 server. Place datafiles and archive redo log files if archiving is enabled, on the NAS filer. Placing the most frequently updated online redo log files on the server helps performance but complicates backup and restore. Media recovery has to be performed twice. Once on the NAS filer, once on the server. You must run NTP to keep time in sync between the server and the NAS filer.

- Management Oriented – Place the instance initialization files (pfiles) on the server. Place the control files, online redo log files, data files, archive redo log files if archiving is enabled, process log files, user log files on the NAS filer. This approach simplifies back and restore. Media recovery needs to be performed only once on the NAS filer. NTP is high recommended to keep time in sync between the NAS and the servers.

- Balanced Approach – Place the instance initialization files (pfiles), process log files, user log files on the server. Place data files, online redo log files, and archive redo log files if archiving is enabled, on the NAS filer. Media recovery can be performed only once on the NAS filer without restoring process log files and user log files on the server. DBA needs to be aware of the differences each media recovery will make in his user log files and process log files. DBA also needs to regularly back up those log files in order to prevent them to overflow the limited disk space on the server. NTP is strongly recommended to keep time in sync between the NAS and the servers.

Oracle Storage Compatibility Program (OSCP)

In response to increasing adoption of NAS as Oracle database storage option, Oracle instituted Oracle Storage Compatibility Program in 1999 for NAS vendors and the like to comply with Oracle standards. OSCP is an important compliance document to know about when implementing NAS. OSCP let you know what Oracle expects from NAS or if your NAS implementation lives up Oracle's expectations. When problems occur this knowledge will assist you in your problem identification and resolution process.

OSCP specifies six areas for compliance. These six areas are

- Synchronous writes. All file caching must be write-through. When a write issued by an Oracle database returns to Oracle, the written data must already be on persistent media. This is critical for the consistency of the Oracle database

- Hard-mount/soft-mount. One can hard mount or soft mount NFS. Both works with Oracle database, with different error behaviors.
    - Comment: With NAS, the best practice is for NFS client to use hard mount. Please refer to Server Requirement - NFS Client Mount Options on page 13 for explanation

- NVRAM. NVRAM rarely fails. But when it fails, the storage system must behave so that (1) Oracle database never returns incorrect data to the user, and (2) one can always recover the database to a consistent state
    - Comment: Since NVRAM works as a hardware cache for disk reads and disk writes and it can produce noticeable performance improvement for a NAS device, many NAS vendors implement NVRAM in their products. Both NetApp 880 and HP NAS 8000 are of no exception in using NVRAM. NVRAM failure is often caused by a run-down battery. Check with the NAS vendor to be fully aware of the characteristics of NVRAM used in your NAS device and the repair procedure when NVRAM does go down

- File locks. Oracle uses file locks to prevent user accidentally starting up a second instance on the same node (in a single instance configuration). Thus network storage system must implement file locks properly according to NFS specifications.

- Double failures. The network storage system must be able to survive double failures. For example, failure of the NFS server, followed by reboot of the NFS server while NFS client fails.

- Stale cache. One common way to improve availability of Oracle database is using the “fail safe” configuration. When one node running a single Oracle instance fails, one immediately starts up another single Oracle instance on a different node. The NFS/Storage system must make sure that second instance never see the stale data.

Appendix D provides a full description of how to test compliance with NAS in the above-listed areas.

High Availability for NAS

Along with the benefits that NAS brings, comes an issue that cannot be ignored for any mission critical systems: NAS reliability. What impact will NAS make on its clients when a NAS filer goes down?

The importance of this issue can be illustrated in the context of our NAS landscape.

Let’s decide the following:

1. The landscape relies upon two NAS filers for storage. One filer supports the file systems for PROD, the other filer supports for DEVL and QA
2. When no system goes down and is running in good working order in the landscape, the landscape is regarded to have 100% system availability to users.
3. When PROD is down, the landscape is considered to have lost 40% of system availability.
4. When either DELV or QA system is down, the landscape is thought to have lost 30% availability.

Table 7 shows system availability in different scenarios.

Table 7 – NAS Landscape System Availability Matrix

System	**State**	**PROD**	**DEVL**	**QA**	**% Resources available**
Filer1 (NetApp F880)	Down	Down	Up	Up	60%
Filer2 (NetApp F880)	Down	Up	Down	Down	40%
Filer1 (NetApp F880) and Filer2 (NAS 8000)	Down	Down	Down	Down	0%
PROD	Down	Down	Up	Up	60%
DEVL	Down	Up	Down	Up	70%
QA	Down	Up	Up	Down	70%

Average (when one system is down)					60% (excluding both filers are down)

While a DAS landscape availability based on comparable presumptions looks like this:

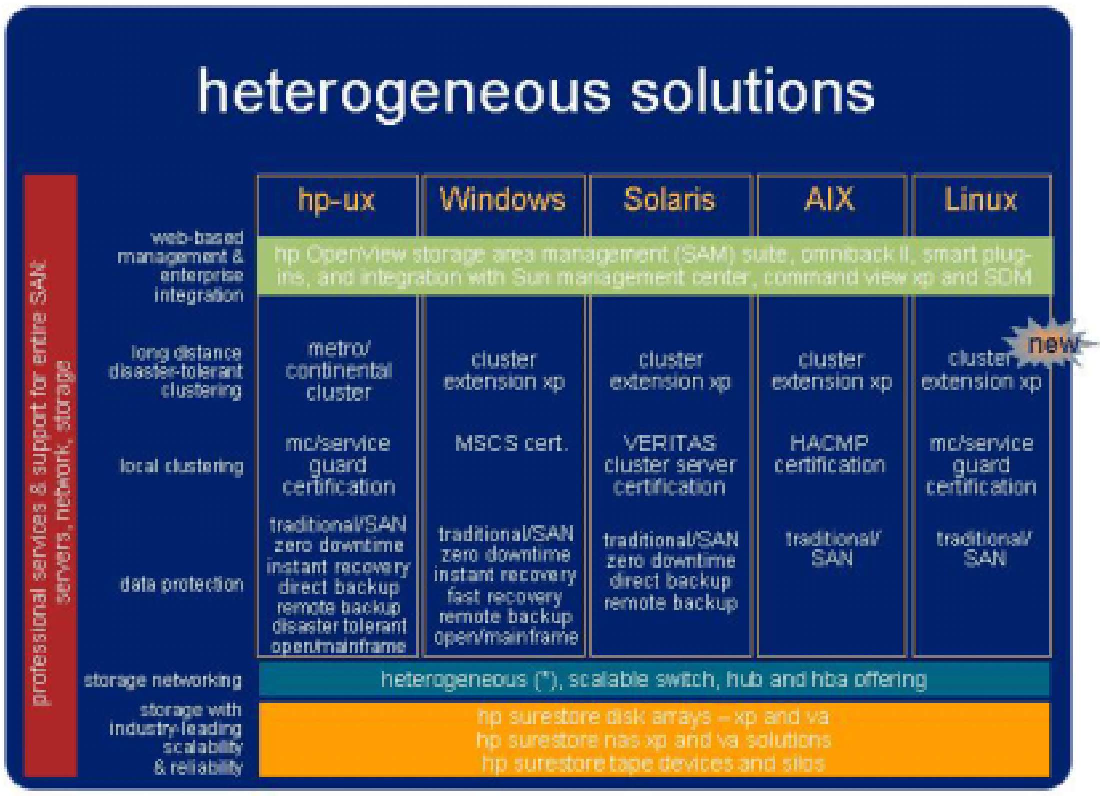
Table 8 – DAS Landscape System Availability Matrix

System	**State**	**PROD**	**DEVL**	**QA**	**% Resources Available**
PROD	Down	Down	Up	Up	60%
DEVL	Down	Up	Down	Up	70%
QA	Down	Up	Up	Down	70%
Average (when one system is down)					67%

As the tables reveal, the NAS-based landscape is more prone to loosing system availability than a DAS landscape. The more you centralize the storage on NAS, the more vulnerable the landscape will become.

One way to address the NAS availability issue is to make NAS highly available.

Table 9 lists a number of High Availability (HA) solutions available today.



heterogeneous solutions

professional services & support for entire SAN: servers, network, storage

	hp-ux	Windows	Solaris	AIX	Linux
web-based management & enterprise integration	hp OpenView storage area management (SAM) suite, omniback II, smart plug-ins, and integration with Sun management center, command view xp and SDM				
long distance disaster-tolerant clustering	metro/ continental cluster	cluster extension xp	cluster extension xp	cluster extension xp	new cluster extension xp
local clustering	mc/service guard certification	MSCS cert.	VERITAS cluster server certification	HACMP certification	mc/service guard certification
data protection	traditional/SAN zero downtime instant recovery direct backup remote backup disaster tolerant open/mainframe	traditional/SAN zero downtime instant recovery fast recovery remote backup open/mainframe	traditional/SAN zero downtime direct backup remote backup	traditional/ SAN	traditional/ SAN
storage networking	heterogeneous (*), scalable switch, hub and hba offering				
storage with industry-leading scalability & reliability	hp surestore disk arrays – xp and va hp surestore nas xp and va solutions hp surestore tape devices and silos				

Source: Business Continuity Solutions, HP Invent, Network Storage Solutions Organization, March 2002

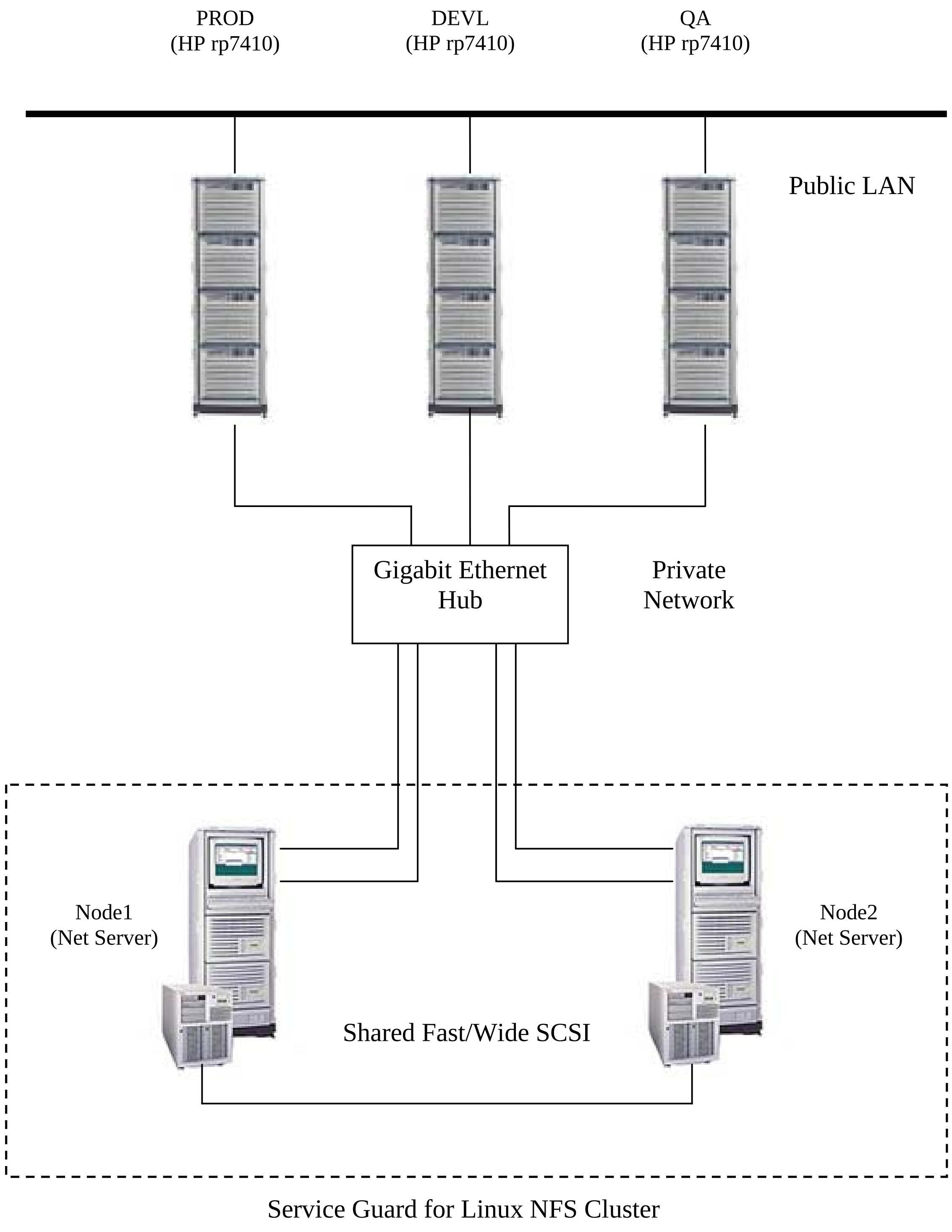
HP MC Service Guard for Linux (SG) and MC Service Guard NFS for Linux (SGN) (not listed in the table) can be used to build a highly available NAS.

SG and SGN offer a number of advantages over other solutions:

- The software runs on standard IA hardware. No particular commercial NAS devices are required.
- The software has HP support. Her sister, MC/SG for HP-UX, is a time-proven, successful HA solution.
- SG for Linux scales to 2 or 4 nodes, depending on the disk technology used in the HA cluster, flexible enough to satisfy most NAS requirement

Figure 4 shows a revised landscape using a HA NAS cluster built with two HP Net Servers, SG, and SGN. The reason to replace NetApp F880 and HP NAS 8000 with Net Servers as NAS storage is to highlight the Service Guard's openness to standard hardware. You can even replace HP Net Servers with any other comparable Linux computers without relying upon any particular vendor. That's the beauty of SG for Linux. That's the beauty of software being able to run on open standard hardware.

Figure 4 - Landscape using Service Guard for Linux Cluster



Before we go further, an understanding of what SG is and how it works is in order.

Service Guard for Linux

In network terms, there are three strategies to implement HA:

- IP Failover
- MAC Failover
- Dynamic DNS Configuration

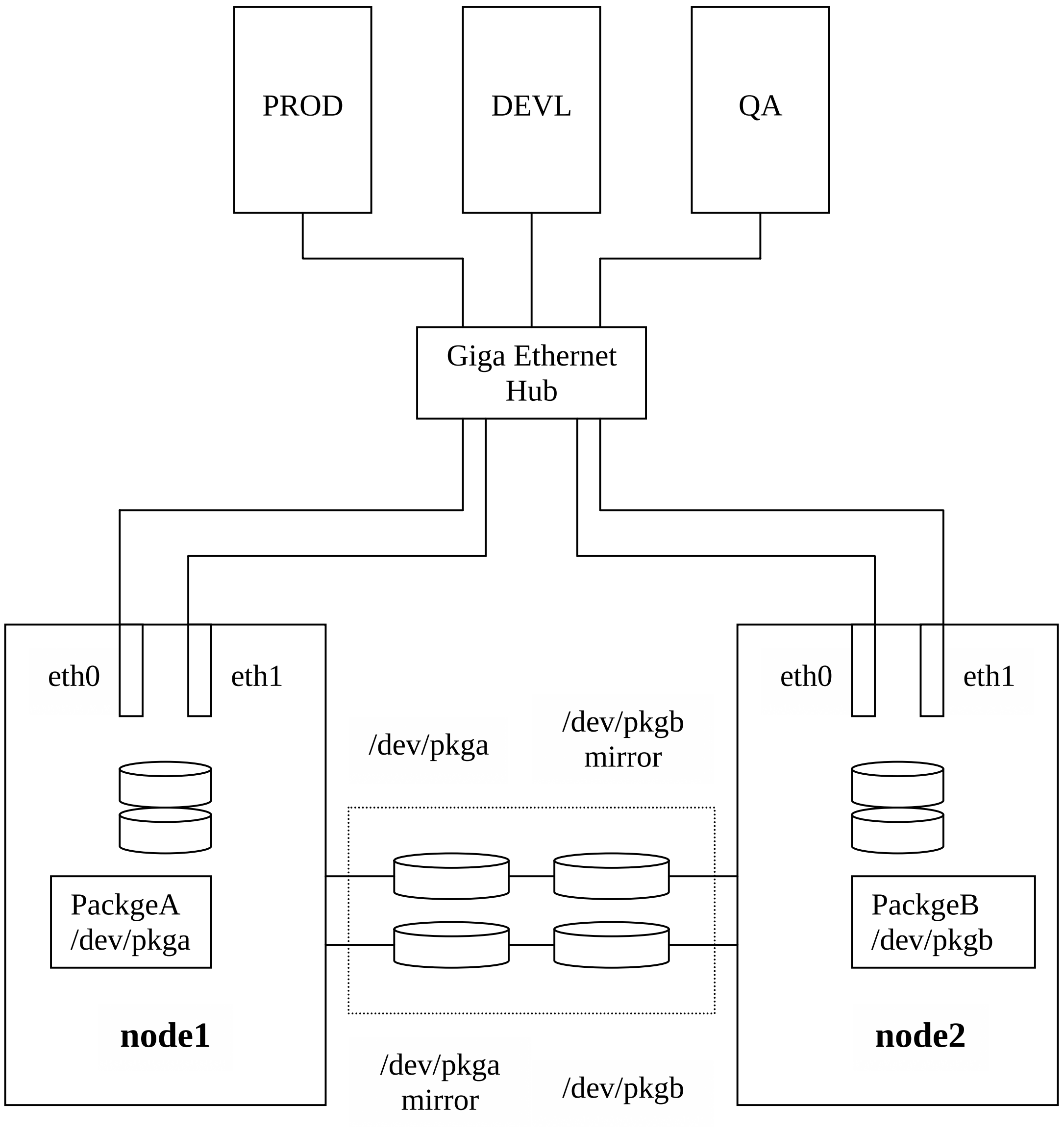
SG for Linux is an HA using IP failover approach. It allows you to create high availability clusters using HP Net Servers and other Intel Architecture (IA) based computer systems running Red Hat Linux 7.1 or above.

A high availability computer system allows application services to continue in spite of a hardware or software failure. High availability systems protect users from software failures as well as from failure of a system processing unit (SPU), disk, or local area network component. In the event that one component fails, the redundant component takes over. SG and other high availability subsystems coordinate the transfer of services between components.

A SG cluster is a networked grouping of HP Net servers known as **nodes.** The cluster has sufficient redundancy of software and hardware so that a single point of failure will not significantly disrupt the services that the cluster provides to users. Application services (individual Linux processes, including NFS) are grouped together in **packages**. In the event of a single service, node, network, or other resource failure, SG will automatically transfer control of the package to another node within the cluster, allowing services to continue with minimal interruption.

Figure 5 shows a more detailed diagram of the SG cluster used in the landscape:

Figure 5 – SG Configuration Diagram



In the cluster, SG provides high availability through SPU redundancy, LAN redundancy and disk redundancy.

<u>SPU Redundancy</u>

The cluster includes two systems identified as Node1 and Node2. Node1 is assigned to run PackageA, which basically provide NFS support for

PROD. Node1 is the Primary Node for PackageA. When Node1 fails, the control for PackageA is transferred to Node2. Node2 is Adoptive Node for PackageA. Node2 is assigned to run PackageB, providing NFS support for DEVL and QA. Node2 is Primary Node for PackageB. When Node2 fails, the control of PackageB is transferred to Node1. Node1 is Adoptive Node for PackageB. This service package transfer relationship between Node1 and Node2 is defined as “**mutual failover**”. Other relationships or “**policies**” exist. They can be found in Appendix E.

<u>LAN Redundancy</u>

Each node has two network interfaces, eth0 and eth1. eth0 is an active interface with a designated IP address bound to it, eth1 is a standby. When both nodes are up and running in the cluster, IP address on eth0 in Node1 is 192.168.0.1. eth1 is not active. Eth0 in Node2 has IP address 192.168.0.2. eth1 in Node2 is not active. PROD accesses PackageA running on Node1 via IP 192.168.0.1; DEVL and QA access PackageB running on Node2 via IP 192.168.0.2.

When Node1 in the cluster fails, the failure triggers the failover process. In the process, Node2 adopts IP 192.168.0.1 from Node1 onto its interface eth1. The control of PackageA is transferred to Node2 and continues to run on Node2. After failover, both eth0 and eth1 interfaces are active on Node2 and both PackageA and PackgeB are running on Node2. Resultantly, PROD will able to continue using PackageA via the same IP address 192.168.0.1 without knowing it is now bound to eth1 on Node2. No impact takes place on DEVL and QA that use PackageB.

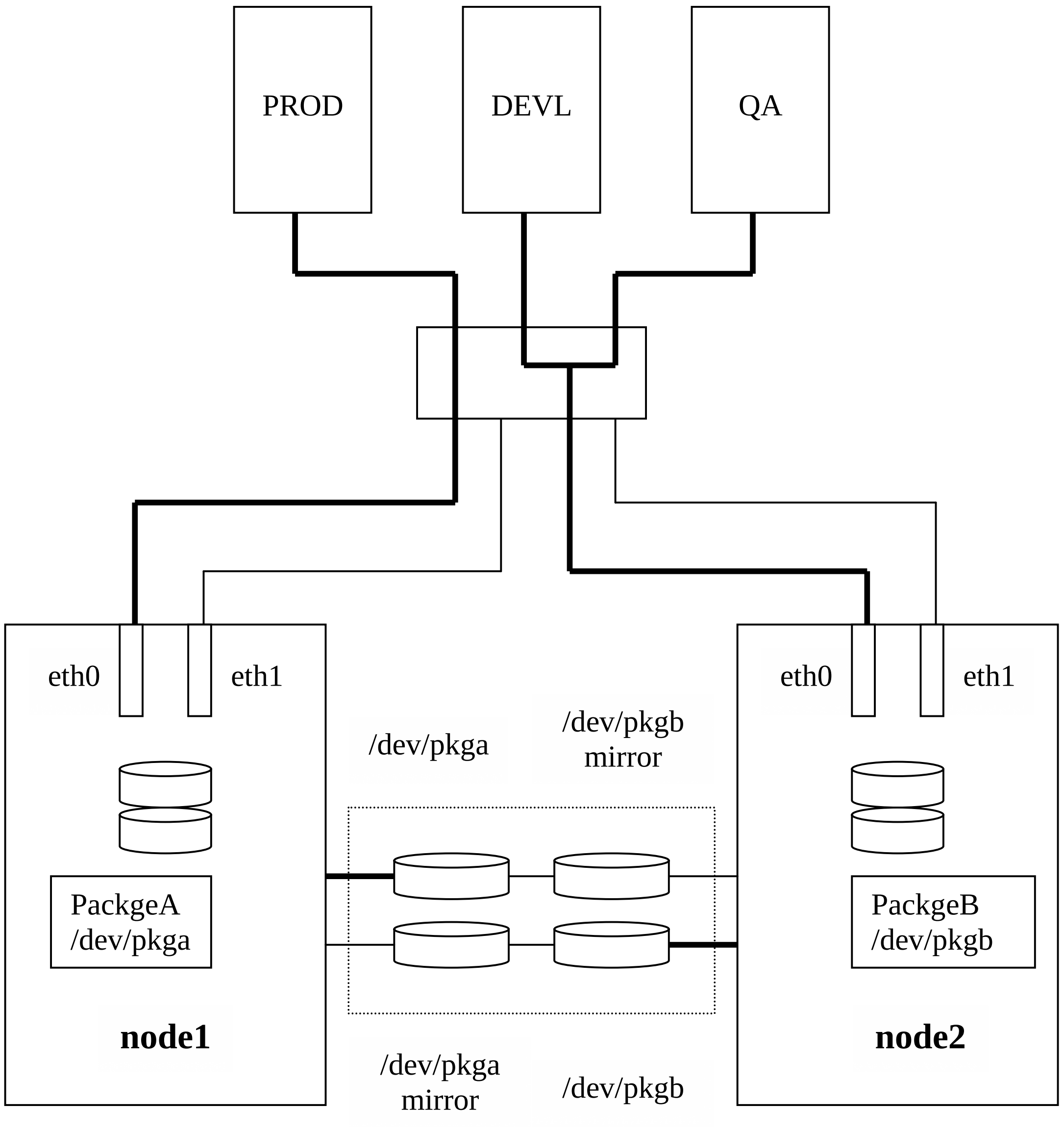
When Node2 fails, PackageB gets transferred to Node1 in the same fashion as described above. The end result is that DEVL and QA continue using PackageB without knowing it is running on Node1. PROD is not affected by the change at all.

<u>Disk Redundancy</u>

The disks are shared between the two nodes. Two sets of volume groups are created using Linux Logical Volume Manager on the shared disks. One set includes one volume group /dev/pkga, the other set includes two volume groups /dev/pkgb_devl and /dev/pkgb_qa. Only one node can access a given set of volume groups at a time. When one node fails, the surving node will access the volume groups supported by the failed node and continue providing the file system services with the volume groups it has adopted. In addition, each volume group is mirrored for data protection in the event of disk failure. The shared storage subsystem configuration is to be discussed in more detail later.

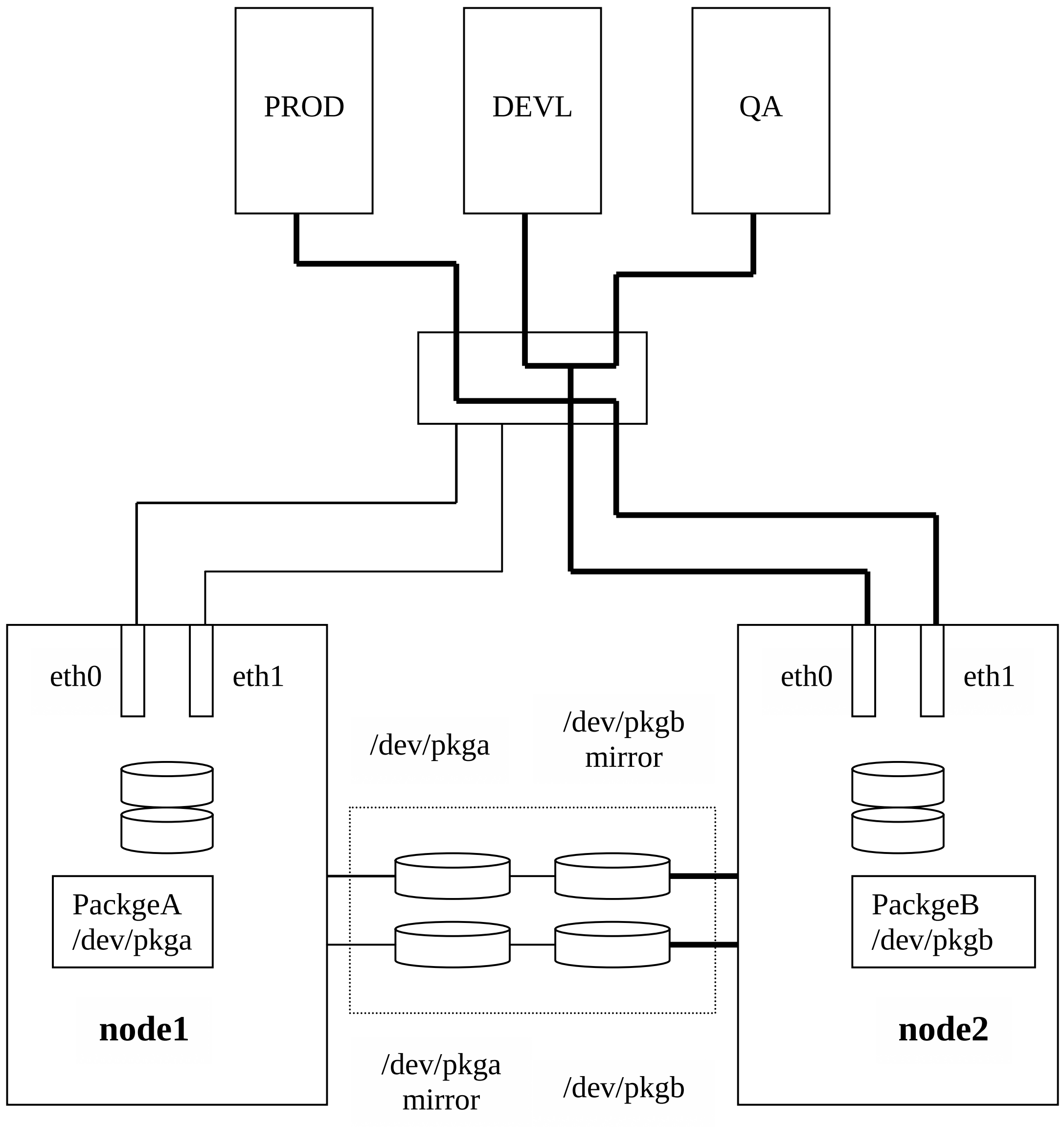
The changes that occur during the cluster failover process are shown in Figure 6 and Figure 7.

Figure 6 - Before Failover



The solid lines in the diagram indicate active interconnects.

Figure 7 – After Failover



The solid lines in the diagram indicate active interconnects.

For a complete reference to SG for Linux, please consult with Managing MC/ServiceGuard for Linux by HP, Manufacturing Part Number: B9903-90005.

Shared Disk Subsystem Configuration for SG

SG for Linux requires use of Linux Logical Volume Manager (LVM) to build required shared disk subsystem.

Linux LVM is very similar to HP-UX LVM. Table 10 compares a list of commonly used commands to define LVM devices under Linux and HP-UX.

Table 10 - LVM Commands

Command	Function	Linux	HP-UX
pvcreate	Prepare disk device	Y	Y
vgcreate	Create volume group	Y	Y
Vgscan	List volume groups	Y	Y
lvcreate	Create logical volume	Y	Y
vgchange	Activate/deactivate a volume group	Y	Y
vgextend	Add new disks to existing volume groups	Y	Y
lvextend	Extend logical volumes	Y	Y
vgcfgbackup	Backup volume group configuration	Y	Y
vgcfgrestore	Restore volume group configuration	Y	Y

One difference between Linux LVM and HP-UX LVM is in implementing mirroring or RAID 1.

Under HP-UX, LVM mirroring is typically done by lvextend –m, which requires MirrorDisk/UX module to be installed on your system. With lvextend –m, you implement LVM mirroring between logical volumes within a volume group. Mirroring between volume groups is not possible. If other levels of RAID are desired, HP Auto Array or comparables are required.

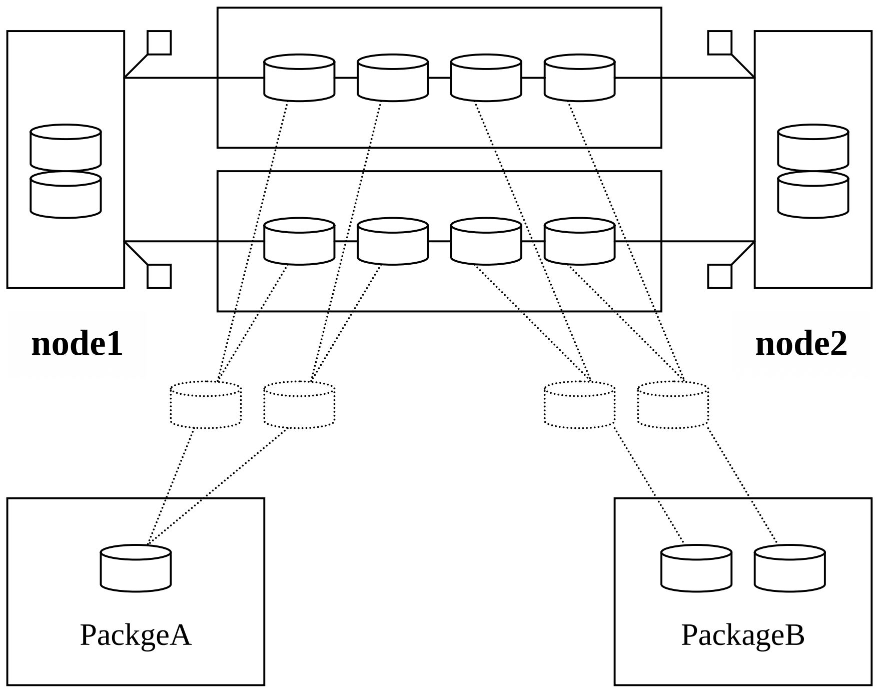
Under Linux, you use Software RAID utility, also called Multiple Device (md) utility, to create mirroring. Mirroring is typically implemented between volume groups instead of logical volumes. In addition to RAID 1, the utility also allows you to create RAID 0, 4, and 5. Software RAID is included in Red Hat Linux 7.1 and above. You can also obtain the package from most Linux web sites.

To better illustrate how to construct mirroring using Linux LVM, we use two JBOD disk systems such as HP Disk System 2300 to build our shared disk subsystem. HP DS 2300 features and specifications can be found in Appendix F.

We populate each JBOD with four (4) 36GB 10K rpm disks on one bus. The two JBOD disk systems are connected to Node1 and Node2 using separate SCSI channels. Each channel is terminated with Y-cable at each end to form a multiple initiator SCSI bus.

Figure 8 below shows the configuration:

Figure 8 – SG Shared Disk Subsystem Configuration Diagram



In the diagram, a total of 8 disks are shared between Node1 and Node2. Those disk devices are:

SCSI Channel 1: /dev/sdc, /dev/sdd, /dev/sde, /dev/sdf
SCSI Channel 2: /dev/sdg, /dev/sdh, /dev/sdi, /dev/sdj

The disks on two channels are mirrored as:

/dev/md0 = /dev/sdc, dev/sdg
/dev/md1 = /dev/sdd, /dev/sdh

/dev/md2 = /dev/sde, /dev/sdi
/dev/md3 = /dev/sdf, /dev/sdj

Three volume groups are constructed using the above four mirrored devices. The three VGs are:

/dev/pkga_vg
/dev/pkgb_vg1
/dev/pkgb_vg2

Three logical volumes are defined in each VG:

/dev/pkga_vg/lvol1 (44GB)
/dev/pkga_vg/lvol2 (8GB)
/dev/pkga_vg/lvol3 (8GB)
/dev/pkga_vg/lvol4 (8GB)
/dev/pkga_vg/lvol5 (4GB)

/dev/pkgb_vg1/lvol1 (22GB)
/dev/pkgb_vg1/lvol2 (4GB)
/dev/pkgb_vg1/lvol3 (4GB)
/dev/pkgb_vg1/lvol4 (4GB)
/dev/pkgb_vg1/lvol5 (2GB)

/dev/pkgb_vg2/lvol1 (22GB)
/dev/pkgb_vg2/lvol2 (4GB)
/dev/pkgb_vg2/lvol3 (4GB)
/dev/pkgb_vg2/lvol4 (4GB)
/dev/pkgb_vg2/lvol5 (2GB)

Furthermore, 15 file systems are created on the LVs listed above. Those file systems and mount points are:

```
node1 # cat /etc/fstab
...
/dev/pkga_vg/lvol1 /vol/prod/data
/dev/pkga_vg/lvol2 /vol/prod/index
/dev/pkga_vg/lvol3 /vol/prod/log
/dev/pkga_vg/lvol4 /vol/prod/tmp
/dev/pkga_vg/lvol5 /vol/prod/adm

/dev/pkgb_vg1/lvol1 /vol/devl/data
/dev/pkgb_vg1/lvol2 /vol/devl/index
/dev/pkgb_vg1/lvol3 /vol/devl/log
/dev/pkgb_vg1/lvol4 /vol/devl/tmp
/dev/pkgb_vg1/lvol5 /vol/prod/adm
```

```
/dev/pkgb_vg2/lvol1 /vol/qa/data
/dev/pkgb_vg2/lvol2 /vol/qa/index
/dev/pkgb_vg2/lvol3 /vol/qa/log
/dev/pkgb_vg2/lvol4 /vol/qa/tmp
/dev/pkga_vg2/lvol5 /vol/prod/adm
```

On NFS clients, the NFS mount file systems are:

On PROD:

```
prod # cat /etc/fstab
...
node1:/vol/prod/data          /prod/data
node1:/vol/prod/index         /prod/index
node1:/vol/prod/log           /prod/log
node1:/vol/prod/tmp           /prod/tmp
node1:/vol/prod/adm           /prod/adm
```

On DEVL:

```
devl # cat /etc/fstab
...
node2:/vol/devl/data          /devl/data
node2:/vol/devl/index         /devl/index
node2:/vol/devl/log           /devl/log
node2:/vol/devl/tmp           /devl/tmp
node2:/vol/devl/adm           /prod/adm
```

On QA:

```
qa # cat /etc/fstab
...
node2:/vol/devl/data          /devl/data
node2:/vol/devl/index         /devl/index
node2:/vol/devl/log           /devl/log
node2:/vol/devl/tmp           /devl/tmp
node2:/vol/devl/adm           /prod/adm
```

Note: The mount options are omitted for clarity.

To achieve the configuration illustrated as above, a certain procedure needs to be followed.

1) Proceed on one node
2) Partition the disks
3) Verify physical volumes

4) Make md devices that mirror selected physical volumes in consideration of load balancing
5) Construct volume groups with appropriate md devices
6) Create logical volumes within each volume group
7) Create file system on each logical volume
8) Mount file system and test the file system’s integrity
9) Unmount file system and deactivate volume groups
10) Back up VG configuration and copy the configuration information to the other node
11) Back up file systems definition (/etc/fstab) and copy it to the other node
12) Import VG configuration
13) Activate shared VGs
14) Mount file systems
15) Test access to shared VGs
16) Deactivate shared VGs

Further details of how to execute the procedure can be found in Appendix G

Thus far, we have built the shared disk subsystem infrastructure required by SG for Linux.

ServiceGuard NFS for Linux

With SG cluster put in place, we can install SGN to make the cluster a HA NAS.

As an add-on to SG for Linux, SGN consists of a set of separate shell scripts and one binary file, providing the following functions on a SG cluster platform:

1. Group NFS server services into packages
2. Define primary node and adoptive node for each NFS package
3. Define failover relationship, i.e., mutual failover, multiple packages failover, or cascading failover
4. Monitors the health of the NFS services on each node in the cluster
5. When a node fails, SG NFS on the node stops NFS services and transfers the control of the NFS package(s) to an adoptive node. The halt process includes but not limited to:
    a. Unmounts file systems
    b. Deactivates volume group or volume groups associated with the NFS pakcage
    c. Stops NFS server services and assists transfer relocatable IP address to the adopitive node
6. On the adoptive node, it starts NFS services associated with the transfered NFS package. The start process includes but not limited to:
    a. Activate volume group or volume groups associated with the package

b. Mount file systems
c. Export file systems
d. Assigns the adopted package IP address to a LAN interface
e. Synchronizes remote mount table
f. Starts providing NFS services for the transferred package

SGN installation procedure can be found in Appendix G.

Service Guard NFS for Linux Configuration

SGN is configured by editing configuration files deployed on both SG cluster nodes. Table 11 lists the configuration files.

Table 11 – SGN Configuration Files

File	Default File Name	Sample Name	Path
Package Configuration File	pkg.conf	pkga.conf	/usr/local/cmcluster/pkga/pkga.conf
		pkgb.conf	/usr/local/cmcluster/pkgb/pkgb.conf
Package Control Script	pkg.cntl	pkga.cntl	/usr/local/cmcluster/pkga/pkga.cntl
		pkgb.cntl	/usr/local/cmcluster/pkgb/pkgb.cntl
NFS Control Script	hanfs.sh	hanfs.sh	/usr/local/cmcluster/bin/hanfs.sh

A close examination of each file as configured to meet our SG Cluster NAS requirement follows. Please note that only sections of each file that are pertaining to those desired, user-definable parameters are listed.

pkga.conf (PackageA, on both nodes)

PACKAGE_NAME	**pkga**
PACKAGE_TYPE	FAILOVER
FAILOVER_POLICY	CONFIGURED_NODE
FAILBACK_POLICY	MANUAL
NODE_NAME	**node1**
NODE_NAME	**node2**
AUTO_RUN	YES
NODE_FAIL_FAST_ENABLED	NO
RUN_SCRIPT	/usr/local/cluster/pkga**/pkga.cntl**
HALT_SCRIPT	/usr/local/cluster/pkga**/pkga.cntl**
SERVICE_NAME	**nfsa.monitor**
SUBNET	192.168.0.0

pkga.cntl (PackageA, on both nodes)

```
PATH=/sbin:/usr/bin:/usr/sbin:/etc:/bin:/usr/local/cmcluster/bin

RAIDTAB=”/usr/local/cmcluster/conf/raidtab.sg
RAIDSTART=”raidstart –c ${RAIDTAB}”
RAIDSTOP=”raidstop –c ${RAIDSTOP}”
VGCHANGE=”vgchange –a y”

MD[0]=/dev/md0
MD[1]=/dev/md1
VG[0]=/dev/pkga_vg

LV[0]=/dev/pkga_vg/lvol1; FS[0]=/vol/prod/data;
FS_MOUNT_OPT[0]=""
LV[1]=/dev/pkga_vg/lvol2; FS[1]=/vol/prod/index;
FS_MOUNT_OPT[1]=""
LV[2]=/dev/pkga_vg/lvol3; FS[2]=/vol/prod/log;
FS_MOUNT_OPT[2]=""
LV[3]=/dev/pkga_vg/lvol4; FS[3]=/vol/prod/tmp;
FS_MOUNT_OPT[3]=""
LV[4]=/dev/pkga_vg/lvol5; FS[4]=/vol/prod/adm;
FS_MOUNT_OPT[4]=""

IP[0]=192.168.0.1
SUBNET[0]=192.168.0.0

HA_NFS_SERVER=”yes”
```

pkgb.conf (PackageB, on both nodes)

```
PACKAGE_NAME              pkgb
PACKAGE_TYPE              FAILOVER
FAILOVER_POLICY           CONFIGURED_NODE
FAILBACK_POLICY           MANUAL
NODE_NAME                 node2
NODE_NAME                 node1
AUTO_RUN                  YES
NODE_FAIL_FAST_ENABLED    NO
RUN_SCRIPT                /usr/local/cluster/pkga/pkgb.cntl
HALT_SCRIPT               /usr/local/cluster/pkga/pkgb.cntl
SERVICE_NAME              nfsb.monitor
SUBNET                    192.168.0.0
```

pkgb.cntl (PackageB, on both nodes)

```
PATH=/sbin:/usr/bin:/usr/sbin:/etc:/bin:/usr/local/cmcluster/bin

RAIDTAB=”/usr/local/cmcluster/conf/raidtab.sg
RAIDSTART=”raidstart –c ${RAIDTAB}”
RAIDSTOP=”raidstop –c ${RAIDSTOP}”
VGCHANGE=”vgchange –a y”

MD[0]=/dev/md2
MD[1]=/dev/md3
VG[0]=/dev/pkgb_vg1
VG[1]=/dev/pkgb_vg2

LV[0]=/dev/pkgb_vg1/lvol1; FS[0]=/vol/devl/data;
FS_MOUNT_OPT[0]=""
LV[1]=/dev/pkgb_vg1/lvol2; FS[1]=/vol/devl/index;
FS_MOUNT_OPT[1]=""
LV[2]=/dev/pkgb_vg1/lvol3; FS[2]=/vol/devl/log;
FS_MOUNT_OPT[2]=""
LV[3]=/dev/pkgb_vg1/lvol4; FS[3]=/vol/devl/tmp;
FS_MOUNT_OPT[3]=""
LV[4]=/dev/pkgb_vg1/lvol5; FS[4]=/vol/devladm;
FS_MOUNT_OPT[4]=""

LV[5]=/dev/pkgb_vg2/lvol1; FS[5]=/vol/qa/data;
FS_MOUNT_OPT[0]=""
LV[6]=/dev/pkgb_vg2/lvol2; FS[6]=/vol/qa/index;
FS_MOUNT_OPT[1]=""
LV[7]=/dev/pkgb_vg2/lvol3; FS[7]=/vol/qa/log; FS_MOUNT_OPT[2]=""
LV[8]=/dev/pkgb_vg2/lvol4; FS[8]=/vol/qa/tmp;
FS_MOUNT_OPT[3]=""
LV[9]=/dev/pkgb_vg2/lvol5; FS[9]=/vol/qa/adm;
FS_MOUNT_OPT[4]=""

IP[0]=192.168.0.2
SUBNET[0]=192.168.0.0

HA_NFS_SERVER=”yes”
```

hanfs.sh (/usr/local/cmcluster/bin/hanfs.sh on **Node1**)

```
XFS[0]=“*:/vol/prod/data”
XFS[1]=“*:/vol/prod/index”
XFS[2]=“*:/vol/prod/log”
XFS[3]=“*:/vol/prod/tmp”
```

```
XFS[4]="*:/vol/prod/adm"

NFS_SERVICE_NAME[0]=nfsa.monitor
NFS_SERVICE_CMD[0]=/usr/local/cmcluster/pkga/nfs.mon
```

hanfs.sh (/usr/local/cmcluster/bin/hanfs.sh on **Node2**)

```
XFS[0]="*:/vol/devl/data"
XFS[1]="*:/vol/devl/index"
XFS[2]="*:/vol/devl/log"
XFS[3]="*:/vol/devl/tmp"
XFS[4]="*:/vol/devl/adm"

XFS[5]="*:/vol/qa/data"
XFS[6]="*:/vol/qa/index"
XFS[7]="*:/vol/qa/log"
XFS[8]="*:/vol/qa/tmp"
XFS[9]="*:/vol/qa/adm"

NFS_SERVICE_NAME[0]=nfsb.monitor
NFS_SERVICE_CMD[0]=/usr/local/cmcluster/pkgb/nfs.mon
```

With DNS using local files, /etc/hosts on both node1 and node2 need to have these entries:

```
…
192.168.0.1    node1
192.168.0.2    node2
…
```

Accordingly, /etc/nsswitch.conf file on both nodes needs to contain this entry:

```
…
hosts:          files nisplus nis dns
…
```

The entire SGN configuration is summarized in Table 12.

Table 12 – SGN Configuration Summary

What is configured	File	Specifics
Failover logic	Package Configuration File	▪ Package name ▪ Failover type ▪ Primary node, adoptive node(s) ▪ Failover and fallback policies, etc.
Shared disk storage access control	Package Control Script	▪ Define volume groups, logical volumes, and file systems associated with a package. ▪ Activates volumes and mounts file systems when starting a package; ▪ Deactivates volume groups and unmounts file systems when stopping a package
Exporting file systems	NFS Control script	▪ Define file systems to be exported. ▪ Specify whether NFS monitor is used

References to more details about how to configure SGN can be found in Appendix G.

As a recap, thanks to MC/ServiceGuard for Linux and MC/ServiceGuard NFS for Linux from HP, we can not only make NAS highly available to address reliability concerns for mission critical systems but also build an High Availability NAS cluster with standard hardware, avoiding the high cost that is often associated with HA solutions based on proprietary hardware.

High Availability for Landscape

In the previous section, we discussed the feasibility of building a highly available NAS for our landscape.

As ServiceGuard NFS for Linux being able to extend ServiceGuard for Linux in NFS services, Oracle 9i Real Application Clusters (ORAC) supports ServiceGuard for HP-UX and is able to extend the HA cluster in providing highly available database services.

Different from SGN, however, ORAC does lot more by introducing a whole new dimension of HA for database services.

Let’s set certain parameters for our landscape before exploring what ORAC can do in making the database services highly available.

1. We decide to cluster the three rp7410 servers with SG (for HP-UX). The three servers become node1, node2, and node3 in the cluster.
2. We wrap the three Oracle instances PROD, DEVL, and QA in three packages.
3. We define multiple package failover policy between the nodes
4. We add two more network interface cards to each node allowing IP failover
5. We use HA NAS that we have just built for the shared storage that HP-UX SG cluster requires [6]

Figure 9 shows the new, double HA enabled landscape.

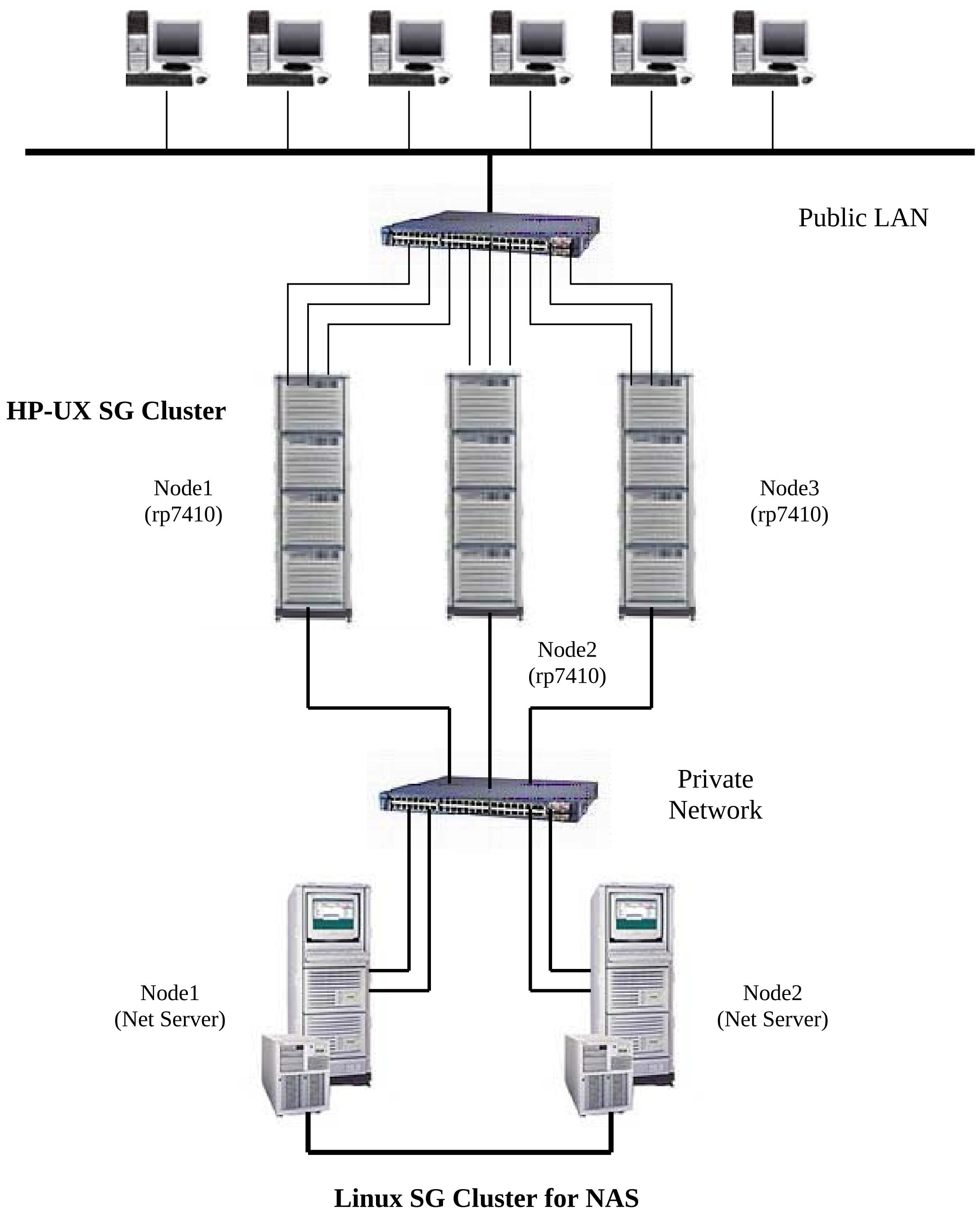
Table 13 tabulates the Oracle instances, packages, and cluster nodes

Table 13- HP-UX SG Cluster Configuration

Instance	Package	Primary Node	Adoptive Node	Adoptive Node
PROD	PKGA	Node1	Node2	Node3
DEVL	PKGB	Node2	Node3	Node1
QA	PKGC	Node3	Node1	Node2

Keep in mind that with all the changes made the landscape still runs three Oracle instances, namely, PROD, DEVL, and QA. No more than one SPU runs one instance in normal circumstances. Instances may run on different nodes determined by TOC when failover occurs. In the worst scenario one SPU has to run three instances when two nodes have failed. For example, when Node1 fails, TOC transfers PackageA (PROD) to Node2 because Node2 is the adoptive node. When Node1 and Node2 both fail, both PackageA and PackageB (DEVL) are transferred to Node3 because Node3 is the adoptive node for both packages. Node3, thus, has to run all three packages including instances of PROD, DEVL, and QA, for which it is the primary node. Just following the logic communicated in the example given and the relationship shown in Table 12 you can figure out what will happen to other packages in other scenarios.

Figure 9 – Double HA Clustered Landscape



When ORAC is used on top of SG, it will form a virtual global database server for each original instance, i.e., PROD, DEVL, or QA using all three cluster nodes. Each virtual server consists of three instances running on three machines simultaneously, resembling Oracle Parallel Server (OPS). This way, ORAC makes full use of all the computing power from the entire cluster. When DEVL and QA are not busy, for instance, PROD will automatically use more of the computing power available from all three nodes instead of one.

More importantly, ORAC introduces Cache Fusion technology that OPS don’t have. What Cache Fusion does is essentially substituting disk I/O with a global cache system in serving as many user requests as possible.

ORAC ‘s ability to capitalize computing power in a cluster and bypass disk I/O is something very desirable and helpful in boosting database performance in a NAS environment.

Now let’s get up and close to see what ORAC is and how it works.

Oracle 9i Real Application Clusters (ORAC)

Oracle 9i supports HA with Oracle Real Applications Clusters (ORAC). ORAC is included in Oracle 9i Enterprise Database edition. According to Oracle, ORAC represents a breakthrough in database high availability technology.

Traditionally, there are three high availability database architectures:

- Federated databases
- Shared-nothing architecture
- Shared-disk architecture

Microsoft SQL Server uses Federated databases approach; IBM DB2 implements Shared-nothing strategy; Oracle Parallel Server (OPS) is based on Shared-disk architecture.

Different from the traditional HA architectures, ORAC use "Shared-Cache" architecture. Cache Fusion is the core of this technology.

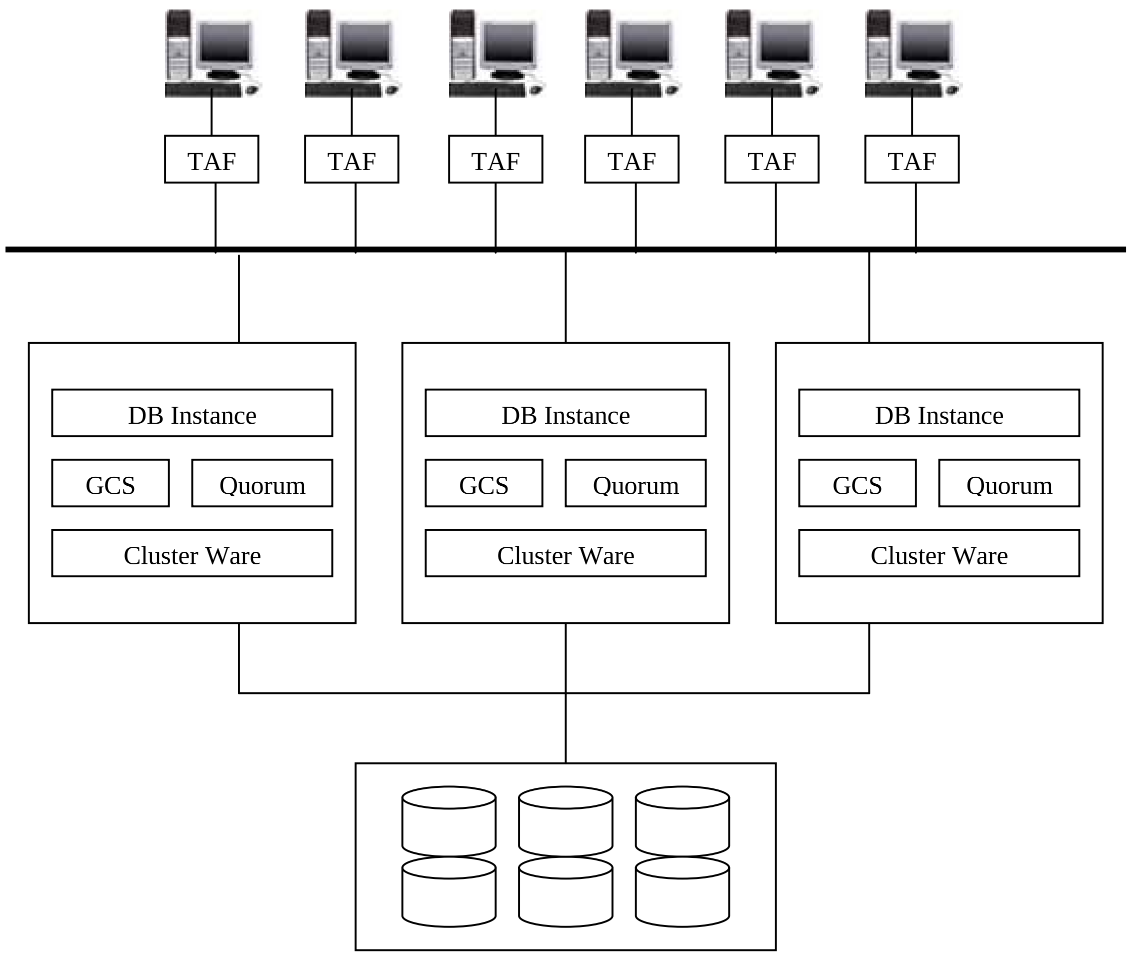
Figure 9 shows ORAC running in a three-node cluster.

ORAC runs in a four-layer model on the server side.

- At the top layer is database instance. Database instance runs on each node.
- In the second layer is the quorum mechanism and GCS (Global Cache Services). Implemented in shared control file, the quorum mechanism determines database failover. Parallel to the quorum mechanism is GCS. What GCS does is mainly sharing cluster-wide caches for data access and coordinating between local caches and remote caches. We'll revisit GCS's role in ORAC in more details shortly.
- In the third layer lies the cluster ware. Oracle supplies its own cluster management software Oracle Cluster Manager for Linux and Windows platforms to do OS failover as well as database failover. In case of HP-UX, it uses MC/ServiceGuard.
- At the bottom is the shared disk subsystem. The subsystem allows the cluster nodes equal access to the storage media and the file systems that the media supports. In here ORAC looks very much like its predecessor Oracle Parallel Server (OPS) as of a shared disk architecture based cluster.

On the client side, TAF (Transparent Application Failover) is used to enhance ORAC load-balance and fail-over. TAF allows each client to have redundant network connections to more than one node. When failover occurs, TAF helps client resume access to services much more quickly than otherwise. For more details about TAF, please find references in Appendix H.

Figure 9 – Three-node Oracle 9i RAC Cluster



Oracle 9i RAC Cache Fusion Technology

Oracle's Cache Fusion architecture is a new shared cache architecture that provides e-business applications the benefits of both shared disk and shared nothing databases without the drawbacks of the either architecture. It does so by exploiting rapidly emerging disk storage and interconnect technologies, including ones that we have been discussing in this paper. While Cache Fusion architecture is revolutionary in an industry sense, it is a natural evolution of the Oracle Parallel Server architecture. See Appendix I about OPS.

Oracle Real Applications Clusters with Cache Fusion architecture provides the following key benefits to enterprise e-business application deployments:

- Flexible and effortless scalability for e-business applications: application users can log onto a single virtual high performance cluster server. Adding nodes to the database is easy and manual intervention is not required to partition data when processor nodes are added or when business requirements change. Cluster scalability for all applications out-of-the box – without modification.

- A high availability solution than traditional cluster database architectures; this architecture provides customers near continuous access to data with minimal interruption by hardware and software component failures. The system is resilient to multiple node failures and component failures are masked from end-users.
- A single management entity; a single system image is preserved across the cluster for all management operations. DBAs perform installation, configuration, backup, upgrade, and monitoring functions once. Oracle then automatically distributes the management functions to the appropriate nodes. This means the DBA manages one virtual server.
- Cache Fusion preserves the investment that all Oracle customers have already made in learning and exploiting Oracle for their e-business applications; all the single node database functionality is preserved and application clients connect to the database using the same standard Oracle application interfaces.

Scalability and performance

E-business application users, or mid tier application server clients, connect to the database by way of a virtual database server name. Oracle automatically balances the user load among the multiple nodes on the cluster. The ORAC database instances on the different nodes subscribe to all or some subset of database services. This provides DBAs the flexibility of choosing whether specific application clients that connect to a particular database service can connect to some or all of the database nodes.]

While each node has a different physical internet protocol address, application clients connect at a logical database service name level. The clients are therefore not concerned with irrelevant issues such as multiple addresses for the multiple server machines.

E-business can painlessly add processing capacity as they grow. The Cache Fusion architecture immediately utilizes the CPU and memory resources of the new node. DBAs do not need to manually re-partition data. This benefit is a by-product of the architecture since transparent data access is fundamental to Cache Fusion.

The Cache Fusion architecture automatically accomplishes the rapidly changing e-business requirements and resulting workload changes. DBAs also do not need to manually re-partition data based on the changed workloads. ORAC dynamically reallocates database resources to optimally utilize the cluster system resources with minimum disk I/O and low latency communication among the nodes. This allows Real Application Clusters to deliver increased application throughput and optimal response time.

In traditional shared disk databases, synchronizing database resources across the cluster database poses significant challenges with regard to achieving cluster scalability. So how does Cache Fusion solve these complex and age-old problems?

Cache Fusion Processing

There are primarily five key breakthroughs that make the Cache Fusion architecture scalable.

- Utilizing the collective (fast) database caches of all the nodes in the system to satisfy application requests to any one node.
- Removing (slow) disk operations from the critical path of inter-node synchronization
- Greatly reducing the required number of messages for inter-node synchronization
- Using scalable inter-node data block transfers that greatly benefit from the above breakthroughs
- Exploiting low latency cluster interconnect protocols for both database messages and data shipping (VIA)

For purposes of discussion, we divide these five breakthroughs into two subtopics:

- The Global Cache Service management of the shared cache
- Efficient inter-node messaging and resource management

Global Cache Service Management of the Shared Cache

Traditional shared disk database system use disk I/O for synchronizing data access across multiple nodes. Typically, when to or more nodes contend for the same data block, the node that has a lock on the data block writes it to disk before the other nodes can access the same data block. This requires disk I/O that involves moving components; it is inherently slow. It also requires synchronization in the form of messages that nodes need to exchange to communicate about lock status. While inter-node message communication can be performed more efficiently with a good design and optimal implementation strategy, the costs of disk I/O for synchronization pose significant challenges for scaling non-partitioned workloads or high contention workloads on traditional shared disk subsystems.

The Cache Fusion architecture overcomes this fundamental weakness in traditional shared disk systems by utilizing the collective caches of all the nodes in the cluster to satisfy database requests. The Global Cache Service manages the status and transfer of data blocks across the buffer caches of the instances. The GCS is tightly integrated with the buffer cache manager to enable fast lookup of resource information in the Global Resource Directory [RAC Resource Coordination]. This directory is distributed across all instances and maintains the status information about resources including any data blocks that require global coordination. Query requests can now be satisfied by the local cache or any of the other caches. This reduces disk I/O. Update operations do not require disk I/O for synchronization since the local node can obtain the needed block directly from any of the cluster database node caches. Expensive disk I/O are only performed when none of the collective caches contain the necessary data and when an update transaction performs a COMMIT operation that requires disk write guarantees. This implementation effectively expands the working set of the database cache and reduces disk I/O to dramatically speed up database operation. No doubt, this cache-based implementation is particularly helpful in improving performance in NAS.

Using our three-node cluster example depicted in Figure 6, let's take a look at four specific types of cross-node contention for data and how Cache Fusion addresses each of them.

1. Read/read – user on node 1 wants to read a block that user on node 2 has recently read
2. Read/write – user on node 1 wants to read a block that user on node 2 has recently updated
3. Write/read – user on node 1 wants to update a block that user on node 2 has recently read
4. Write/write – user on node 1 wants to update a block that user on node 2 has recently updated
5. Consistent read – user on node 1 wants to read a block that user on node 2 has recently read but user on node 3 is updating

The read/read case typically requires little or no coordination, depending the on specific database implementation. In a traditional shared disk implementation the request by the user on node 1 will be satisfied either via local cache access or by way of disk read operations. In the Cache Fusion implementation, the read request may be served by any of the caches in the cluster database where the order of access preference is local cache, remote cache, and finally disk I/O. If the query request is served by a remote cache, the block is transferred across the high speed cluster interconnect from one node's cache to another and expensive disk I/O is avoided.

Both the read/write and write/write cases, in which a user on node 1 updates the block, coordination between the instances become necessary so that the block being read is a read consistent image (for read/write) and the block being updated preserves data integrity (for write/write). In both cases, the node that holds the initially updated data block ships the block to the requesting node across the high speed cluster interconnect and avoids expensive disk I/O for read. It does so this with full recovery capabilities.

In the case of the write/read case, node 1 wants to update a block that's already read and cached by a remote instance or node 2. An update operation typically involves reading the relevant block into memory and then writing the updated block back to disk. In this scenario disk I/O for read is avoided and performance is increased as the block is shipped from the cache of node 2 into the cache of node 1.

In the case of serving a consistent read request from a block that has been recently read by a remote instance and yet is being updated by another remote instance, GCS relies upon assignment of resource modes and roles in the global resource dictionary to coordinate the task.

Dynamic Remastering of Resources

The GCS tightly integrated with the buffer cache enables the database to automatically adapt and migrate resources in the Global Resource Directory to a node that establishes a frequent access pattern for a particular data set. Dynamic resource remastering provides the key benefit of greatly increasing the likelihood of local cache access without additional IPC traffic. This is a dramatic improvement over manual partitioning and provides efficiencies in the Cache Fusion architecture by reducing both the number of resource-related inter-node messages and data block transfers across the cluster interconnect. Dynamic remastering works for data that has clear access patterns established, in other words, a set of data is most often accessed by database clients connecting to a particular node. The buffer cache implements the policy that determines when dynamic remastering should be invoked based on observed access patterns. This improves e-business application response by providing local resource management. Furthermore, it reduces the frequency of inter-node communications for synchronization.

By replacing slow and expensive disk I/O with fast cache access in serving most database requests, ORAC can significantly improve overall performance in a cluster using NAS as shared data storage subsystem.

Conclusion

Through our discussion, we have seen how NAS can be used in an HP landscape as a cost-effective, easy to implement, easy to manage, and easy to scale storage solution. With HP high availability technology and Oracle Real Application Clusters you can make NAS solutions reliable and efficient enough to support most demanding mission critical applications.

In today’s bottom-line economy, enterprises have to gather as much business intelligence as they can, record as much business activities as they can, and analyze market trends using as much historical data as they can. Business operations require immediate and constant access to information. Disaster prevention and recovery require multiple data backups and replication to multiple sites. Data warehouse, data farm, and data mart have to be large enough to serve and flexible enough to adapt to ever changing business needs. All of these activities demand for dynamic data storage. To meet the ever growing data storage demand, buyers have to balance the cost and the quantity and quality. The cost will have to include the cost to operate storage solutions with existing staff resources as much as they can. It is in this environment NAS comes to offer what we want:

- Low cost delivery using open source software and standard hardware
- Easy management by attaching to networks and “plug and play”
- High scalability
- Reasonable performance, which can be enhanced in various ways
- Heterogeneous system support

High availability is one area where NAS will continue to develop and mature. Network Appliances reported two Oracle 9i Real Application Clusters certified solutions with NetApp F880 this year is such an example (See Appendix J for details). R&D resources are being spent in easing NAS network performance bottleneck. Virtual Interface Architecture (VIA) being developed in open source community and Direct Access File System (DAFS) developed by Network Appliances are results of these efforts. Both bypass TCP/IP allowing more direct connect between networked resources.

NAS market is expected to exceed $10 billion in one year or two. We will certainly see rapid growth on NAS applications in HP environment as well.

Appendix A – HP Storage Options for DAS

Product	Type	Description
DS2300	JBOD	http://www.hp.com/products1/storage/products/disk_arrays/disksystems/index.html
DS2405	2GB FC-AL	http://www.hp.com/products1/storage/products/disk_arrays/disksystems/ds2405/index.html
VA7100	Disk Array	http://www.hp.com/products1/storage/products/disk_arrays/midrange/va7100/index.html
VA7400	Disk Array	http://www.hp.com/products1/storage/products/disk_arrays/midrange/va7400/index.html
VA7410	Disk Array	http://www.hp.com/products1/storage/products/disk_arrays/midrange/va7410/index.html

Appendix B – NFS Tuning

NFS Performance Tuning for HPUX 11/11i by Dave Olker, 2001, HP Network Research Lab. - http://docs.hp.com/hpux/onlinedocs/netcom/NFS_perf_tuning_hpux110_11i.pdf

Appendix C – POSTMARK Source Code

```
/*
Written by Jeffrey Katcher under contract to Network Appliance.
Copyright (C) 1997-2001
Network Appliance, Inc.

This code has been successfully compiled and run by Network
Appliance on various platforms, including Solaris 2 on an Ultra-170,
and Windows NT on a Compaq ProLiant. However, this PostMark source
code is distributed under the Artistic License appended to the end
of this file. As such, no support is provided. However, please report
any errors to the author, Jeffrey Katcher <katcher@netcom.com>, or to
Andy Watson <watson@netapp.com>.

Versions:
1.00 - Original release - 8/17/97

1.01 - Fixed endless loop on EOF,
       Divide by zero when file_size_high=file_size_low - 10/29/97
       (Thanks to Chuck Murnane)

1.1 - Added new commands to distribute work across multiple directories
      and/or file systems and multiple work subdirectories.

      Changed set location command (+,-) to allow file systems &
weights
      Added set subdirectories command and code to distribute work
across
         multiple subdirectories
      Added file redirect to show and run commands
      Improved help system - 4/8/98

1.11 - Fixed unfortunate problem where read_file opens in append mode
thus
       avoiding actual reads.  (Thanks to Kent Peacock)

1.12 - Changed bytes read and written to float.  Hopefully this will
avoid
       overflow when very large file sizes are used.

1.13 - Added terse report option allowing results to be easily included
in
       other things.  (Thanks to Walter Wong)
       Also tweaked help code to allow partial matches

1.14 - Automatically stop run if work files are depleted

1.5 - It was pointed out by many (most recently Michael Flaster) that
the
      pseudo-random number generator was more pseudo than random.
After
      a review of the literature and extensive benchmarking, I've
replaced
      the previous PRNG with the Mersenne Twister.  While an excellent
PRNG,
      it retains much of the performance of the previous implementation.
      URL: http://www.math.keio.ac.jp/~matumoto/emt.html
      Also changed MB definition to 1024KB, tweaked show command
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <fcntl.h>

#ifdef _WIN32
#include <io.h>
#include <direct.h>

#define GETWD(x) getcwd(x,MAX_LINE)
#define MKDIR(x) mkdir(x)
#define SEPARATOR "\\"
#else
extern char *getwd();

#define GETWD(x) getwd(x)
#define MKDIR(x) mkdir(x,0700)
#define SEPARATOR "/"
#endif

#define MAX_LINE 255
#define MAX_FILENAME 80

#define KILOBYTE 1024
#define MEGABYTE (KILOBYTE*KILOBYTE)

#define PROMPT "pm>"

typedef struct { /* ADT for table of CLI commands */
   char *name;    /* name of command */
   int (*func)(); /* pointer to callback function */
   char *help;    /* descriptive help string */
} cmd;

extern int cli_set_size();
extern int cli_set_number();
extern int cli_set_seed();
extern int cli_set_transactions();
extern int cli_set_location();
extern int cli_set_subdirs();
extern int cli_set_read();
extern int cli_set_write();
extern int cli_set_buffering();
extern int cli_set_bias_read();
extern int cli_set_bias_create();
extern int cli_set_report();

extern int cli_run();
extern int cli_show();
extern int cli_help();
extern int cli_quit();
```

```
cmd command_list[]={ /* table of CLI commands */
   {"set size",cli_set_size,"Sets low and high bounds of files"},
   {"set number",cli_set_number,"Sets number of simultaneous files"},
   {"set seed",cli_set_seed,"Sets seed for random number generator"},
   {"set transactions",cli_set_transactions,"Sets number of
transactions"},
   {"set location",cli_set_location,"Sets location of working files"},
   {"set subdirectories",cli_set_subdirs,"Sets number of
subdirectories"},
   {"set read",cli_set_read,"Sets read block size"},
   {"set write",cli_set_write,"Sets write block size"},
   {"set buffering",cli_set_buffering,"Sets usage of buffered I/O"},
   {"set bias read",cli_set_bias_read,
      "Sets the chance of choosing read over append"},
   {"set bias create",cli_set_bias_create,
      "Sets the chance of choosing create over delete"},
   {"set report",cli_set_report,"Choose verbose or terse report
format"},
   {"run",cli_run,"Runs one iteration of benchmark"},
   {"show",cli_show,"Displays current configuration"},
   {"help",cli_help,"Prints out available commands"},
   {"quit",cli_quit,"Exit program"},
   NULL
};

extern void verbose_report();
extern void terse_report();
void (*reports[])()={verbose_report,terse_report};

/* Counters */
int files_created;  /* number of files created */
int files_deleted;  /* number of files deleted */
int files_read;     /* number of files read */
int files_appended; /* number of files appended */
float bytes_written; /* number of bytes written to files */
float bytes_read;    /* number of bytes read from files */

/* Configurable Parameters */
int file_size_low=500;
int file_size_high=10000;       /* file size: fixed or random within
range */
int simultaneous=500;           /* simultaneous files */
int seed=42;                    /* random number generator seed */
int transactions=500;           /* number of transactions */
int subdirectories=0;         /* Number of subdirectories */
int read_block_size=512;        /* I/O block sizes */
int write_block_size=512;
int bias_read=5;                /* chance of picking read over append
*/
int bias_create=5;              /* chance of picking create over delete
*/
int buffered_io=1;              /* use C library buffered I/O */
int report=0;                   /* 0=verbose, 1=terse report format */

/* Working Storage */
char *file_source; /* pointer to buffer of random text */
```

```
typedef struct {
   char name[MAX_FILENAME+1]; /* name of individual file */
   int size;                  /* current size of file, 0 = unused file
slot */
} file_entry;

file_entry *file_table; /* table of files in use */
int file_allocated;     /* pointer to last allocated slot in file_table
*/

typedef struct file_system_struct {
   file_entry system;
   struct file_system_struct *next,*prev;
} file_system;

file_system *file_systems; /* table of file systems/directories to use
*/
int file_system_weight;    /* sum of weights for all file systems */
int file_system_count;     /* number of configured file systems */
char **location_index;     /* weighted index of file systems */

char *read_buffer; /* temporary space for reading file data into */

#define RND(x) ((x>0)?(genrand() % (x)):0)
extern unsigned long genrand();
extern void sgenrand();

/* converts integer values to byte/kilobyte/megabyte strings */
char *scale(i)
int i;
{
   static char buffer[MAX_LINE]; /* storage for current conversion */

   if (i/MEGABYTE)
      sprintf(buffer,"%.2f megabytes",(float)i/MEGABYTE);
   else
      if (i/KILOBYTE)
         sprintf(buffer,"%.2f kilobytes",(float)i/KILOBYTE);
      else
         sprintf(buffer,"%d bytes",i);

   return(buffer);
}

/* converts float values to byte/kilobyte/megabyte strings */
char *scalef(i)
float i;
{
   static char buffer[MAX_LINE]; /* storage for current conversion */

   if (i/(float)MEGABYTE>1)
      sprintf(buffer,"%.2f megabytes",i/(float)MEGABYTE);
   else
      if (i/(float)KILOBYTE)
         sprintf(buffer,"%.2f kilobytes",i/(float)KILOBYTE);
      else
```

```
         sprintf(buffer,"%f bytes",i);

   return(buffer);
}

/* UI callback for 'set size' command - sets range of file sizes */
int cli_set_size(param)
char *param; /* remainder of command line */
{
   char *token;
   int size;

   if (param && (size=atoi(param))>0)
      {
      file_size_low=size;
      if ((token=strchr(param,' ')) && (size=atoi(token))>0 &&
         size>=file_size_low)
         file_size_high=size;
      else
         file_size_high=file_size_low;
      }
   else
      fprintf(stderr,"Error: no file size low or high bounds
specified\n");

   return(1);
}

/* UI callback for 'set number' command - sets number of files to
create */
int cli_set_number(param)
char *param; /* remainder of command line */
{
   int size;

   if (param && (size=atoi(param))>0)
      simultaneous=size;
   else
      fprintf(stderr,"Error: no file number specified\n");

   return(1);
}

/* UI callback for 'set seed' command - initial value for random number
gen */
int cli_set_seed(param)
char *param; /* remainder of command line */
{
   int size;

   if (param && (size=atoi(param))>0)
      seed=size;
   else
      fprintf(stderr,"Error: no random number seed specified\n");

   return(1);
}
```

```
/* UI callback for 'set transactions' - configure number of
transactions */
int cli_set_transactions(param)
char *param; /* remainder of command line */
{
   int size;

   if (param && (size=atoi(param))>0)
      transactions=size;
   else
      fprintf(stderr,"Error: no transactions specified\n");

   return(1);
}

int parse_weight(params)
char *params;
{
   int weight=1;
   char *split;

   if (split=strrchr(params,' '))
      {
      *split='\0';
      if ((weight=atoi(split+1))<=0)
         {
         fprintf(stderr,"Error: ignoring invalid weight
'%s'\n",split+1);
         weight=1;
         }
      }

   return(weight);
}

void add_location(params,weight)
char *params;
int weight;
{
   file_system *new_file_system;

   if (new_file_system=(file_system *)calloc(1,sizeof(file_system)))
      {
      strcpy(new_file_system->system.name,params);
      new_file_system->system.size=weight;

      if (file_systems)
         {

         new_file_system->prev=file_systems->prev;
         file_systems->prev->next=new_file_system;
         file_systems->prev=new_file_system;
         }
      else
         {
         new_file_system->prev=new_file_system;
         file_systems=new_file_system;
         }

      file_system_weight+=weight;
      file_system_count++;
      }
}

void delete_location(loc_name)
char *loc_name;
{
   file_system *traverse;

   for (traverse=file_systems; traverse; traverse=traverse->next)
      if (!strcmp(traverse->system.name,loc_name))
         {
         file_system_weight-=traverse->system.size;
         file_system_count--;

         if (file_systems->prev==file_systems)
            {
            free(file_systems);
            file_systems=NULL;
            }
         else
            {
            if (file_systems->prev==traverse)
               file_systems->prev=traverse->prev;

            if (traverse==file_systems)
               file_systems=file_systems->next;
            else
               traverse->prev->next=traverse->next;

            if (traverse->next)
               traverse->next->prev=traverse->prev;

            free(traverse);
            }

         break;
         }

   if (!traverse)
      fprintf(stderr,"Error: cannot find location '%s'\n",loc_name);
}

void delete_locations()
{
   file_system *next;

   while (file_systems)
      {
      next=file_systems->next;
      free(file_systems);
      file_systems=next;
      }
   file_system_weight=0;
   file_system_count=0;
}

/* UI callback for 'set location' - configure current working directory
*/
int cli_set_location(param)
char *param; /* remainder of command line */
{
   if (param)
      {
      switch (*param)
         {
         case '+': /* add location to list */
            add_location(param+1,parse_weight(param+1));
            break;

         case '-': /* remove location from list */
            delete_location(param+1);
            break;

         default:
            delete_locations();
            add_location(param,parse_weight(param));
         }
      }
   else
      fprintf(stderr,"Error: no directory name specified\n");

   return(1);
}

/* UI callback for 'set subdirectories' - configure number of
subdirectories */
int cli_set_subdirs(param)
char *param; /* remainder of command line */
{
   int subdirs;

   if (param && (subdirs=atoi(param))>=0)
      subdirectories=subdirs;
   else
      fprintf(stderr,"Error: invalid number of subdirectories
specified\n");

   return(1);
}

/* UI callback for 'set read' - configure read block size (integer) */
int cli_set_read(param)
char *param; /* remainder of command line */
{
   int size;

   if (param && (size=atoi(param))>0)
      read_block_size=size;
   else
      fprintf(stderr,"Error: no block size specified\n");

   return(1);
}

/* UI callback for 'set write' - configure write block size (integer)
*/
int cli_set_write(param)
char *param; /* remainder of command line */
{
   int size;

   if (param && (size=atoi(param))>0)
      write_block_size=size;
   else
      fprintf(stderr,"Error: no block size specified\n");

   return(1);
}

/* UI callback for 'set buffering' - sets buffering mode on or off
   - true = buffered I/O (default), false = raw I/O */
int cli_set_buffering(param)
char *param; /* remainder of command line */
{
   if (param && (!strcmp(param,"true") || !strcmp(param,"false")))
      buffered_io=(!strcmp(param,"true"))?1:0;
   else
      fprintf(stderr,"Error: no buffering mode (true/false)
specified\n");

   return(1);
}

/* UI callback for 'set bias read' - sets probability of read vs.
append */
int cli_set_bias_read(param)
char *param; /* remainder of command line */
{
   int value;

   if (param && (value=atoi(param))>=-1  && value<=10)
      bias_read=value;
   else
      fprintf(stderr,
        "Error: no bias specified (0-10 for greater chance,-1 to
disable)\n");

   return(1);
}


/* UI callback for 'set bias create' - sets probability of create vs.
delete */
int cli_set_bias_create(param)
char *param; /* remainder of command line */
```

```
{
   int value;

   if (param && (value=atoi(param))>=-1  && value<=10)
      bias_create=value;
   else
      fprintf(stderr,
         "Error: no bias specified (0-10 for greater chance,-1 to
disable)\n");

   return(1);
}

/* UI callback for 'set report' - chooses verbose or terse report
formats */
int cli_set_report(param)
char *param; /* remainder of command line */
{
   int match=0;

   if (param)
      {
      if (!strcmp(param,"verbose"))
         report=0;
      else
         if (!strcmp(param,"terse"))
            report=1;
         else
            match=-1;
      }

   if (!param || match==-1)
      fprintf(stderr,"Error: either 'verbose' or 'terse' required\n");

   return(1);
}

/* populate file source buffer with 'size' bytes of readable randomness
*/
char *initialize_file_source(size)
int size; /* number of bytes of junk to create */
{
   char *new_source;
   int i;

   if ((new_source=(char *)malloc(size))==NULL) /* allocate buffer */
      fprintf(stderr,"Error: failed to allocate source file of size
%d\n",size);
   else
      for (i=0; i<size; i++) /* file buffer with junk */
         new_source[i]=32+RND(95);

   return(new_source);
}

/* returns differences in times -
   1 second is the minimum to avoid divide by zero errors */
```

```
time_t diff_time(t1,t0)
time_t t1;
time_t t0;
{
   return((t1-=t0)?t1:1);
}

/* prints out results from running transactions */
void
verbose_report(fp,end_time,start_time,t_end_time,t_start_time,deleted)
FILE *fp;
time_t end_time,start_time,t_end_time,t_start_time; /* timers from run
*/
int deleted; /* files deleted back-to-back */
{
   time_t elapsed,t_elapsed;
   int interval;

   elapsed=diff_time(end_time,start_time);
   t_elapsed=diff_time(t_end_time,t_start_time);

   fprintf(fp,"Time:\n");
   fprintf(fp,"\t%d seconds total\n",elapsed);
   fprintf(fp,"\t%d seconds of transactions (%d per
second)\n",t_elapsed,
      transactions/t_elapsed);

   fprintf(fp,"\nFiles:\n");
   fprintf(fp,"\t%d created (%d per second)\n",files_created,
      files_created/elapsed);

   interval=diff_time(t_start_time,start_time);
   fprintf(fp,"\t\tCreation alone: %d files (%d per
second)\n",simultaneous,
      simultaneous/interval);
   fprintf(fp,"\t\tMixed with transactions: %d files (%d per second)\n",
      files_created-simultaneous,(files_created-
simultaneous)/t_elapsed);
   fprintf(fp,"\t%d read (%d per
second)\n",files_read,files_read/t_elapsed);
   fprintf(fp,"\t%d appended (%d per second)\n",files_appended,
      files_appended/t_elapsed);
   fprintf(fp,"\t%d deleted (%d per second)\n",files_created,
      files_created/elapsed);

   interval=diff_time(end_time,t_end_time);
   fprintf(fp,"\t\tDeletion alone: %d files (%d per second)\n",deleted,
      deleted/interval);
   fprintf(fp,"\t\tMixed with transactions: %d files (%d per second)\n",
      files_deleted-deleted,(files_deleted-deleted)/t_elapsed);

   fprintf(fp,"\nData:\n");
   fprintf(fp,"\t%s read ",scalef(bytes_read));
   fprintf(fp,"(%s per second)\n",scalef(bytes_read/(float)elapsed));
   fprintf(fp,"\t%s written ",scalef(bytes_written));
   fprintf(fp,"(%s  per  second)\n",scalef(bytes_written/(float)elapsed));
}
```

```
void
terse_report(fp,end_time,start_time,t_end_time,t_start_time,deleted)
FILE *fp;
time_t end_time,start_time,t_end_time,t_start_time; /* timers from run
*/
int deleted; /* files deleted back-to-back */
{
   time_t elapsed,t_elapsed;
   int interval;

   elapsed=diff_time(end_time,start_time);
   t_elapsed=diff_time(t_end_time,t_start_time);
   interval=diff_time(t_start_time,start_time);

   fprintf(fp,"%d %d %.2f ", elapsed, t_elapsed,
      (float)transactions/t_elapsed);
   fprintf(fp, "%.2f %.2f %.2f ", (float)files_created/elapsed,
      (float)simultaneous/interval,
      (float)(files_created-simultaneous)/t_elapsed);
   fprintf(fp, "%.2f %.2f ", (float)files_read/t_elapsed,
      (float)files_appended/t_elapsed);
   fprintf(fp, "%.2f %.2f %.2f ", (float)files_created/elapsed,
      (float)deleted/interval,
      (float)(files_deleted-deleted)/t_elapsed);
   fprintf(fp, "%.2f %.2f\n", (float)bytes_read/elapsed,
      (float)bytes_written/elapsed);
}

/* returns file_table entry of unallocated file
   - if not at end of table, then return next entry
   - else search table for gaps */
int find_free_file()
{
   int i;

   if (file_allocated<simultaneous<<1 &&
file_table[file_allocated].size==0)
      return(file_allocated++);
   else /* search entire table for holes */
      for (i=0; i<simultaneous<<1; i++)
         if (file_table[i].size==0)
            {
            file_allocated=i;
            return(file_allocated++);
            }

   return(-1); /* return -1 only if no free files found */
}

/* write 'size' bytes to file 'fd' using unbuffered I/O */
void write_blocks(fd,size)
int fd;
int size;   /* bytes to write to file */
{
   int offset=0; /* offset into file */
   int i;
   /* write even blocks */
   for (i=size; i>=write_block_size;
      i-=write_block_size,offset+=write_block_size)
      write(fd,file_source+offset,write_block_size);

   write(fd,file_source+offset,i); /* write remainder */

   bytes_written+=size; /* update counter */
}

/* write 'size' bytes to file 'fp' using buffered I/O */
void fwrite_blocks(fp,size)
FILE *fp;
int size;   /* bytes to write to file */
{
   int offset=0; /* offset into file */
   int i;

   /* write even blocks */
   for (i=size; i>=write_block_size;
      i-=write_block_size,offset+=write_block_size)
      fwrite(file_source+offset,write_block_size,1,fp);

   fwrite(file_source+offset,i,1,fp); /* write remainder */

   bytes_written+=size; /* update counter */
}

void create_file_name(dest)
char *dest;
{
   char conversion[MAX_LINE+1];

   *dest='\0';
   if (file_system_count)
      {
      strcat(dest,

location_index[(file_system_count==1)?0:RND(file_system_weight)]);
      strcat(dest,SEPARATOR);
      }

   if (subdirectories>1)
      {
      sprintf(conversion,"s%d%s",RND(subdirectories),SEPARATOR);
      strcat(dest,conversion);
      }

   sprintf(conversion,"%d",++files_created);
   strcat(dest,conversion);
}

/* creates new file of specified length and fills it with data */
void create_file(buffered)
int buffered; /* 1=buffered I/O (default), 0=unbuffered I/O */
{
   FILE *fp=NULL;
   int fd=-1;
   int free_file; /* file_table slot for new file */

   if ((free_file=find_free_file())!=-1) /* if file space is available
*/
      { /* decide on name and initial length */
      create_file_name(file_table[free_file].name);

      file_table[free_file].size=
         file_size_low+RND(file_size_high-file_size_low);

      if (buffered)
         fp=fopen(file_table[free_file].name,"w");
      else
         fd=open(file_table[free_file].name,O_RDWR|O_CREAT,0644);

      if (fp || fd!=-1)
         {
         if (buffered)
            {
            fwrite_blocks(fp,file_table[free_file].size);
            fclose(fp);
            }
         else
            {
            write_blocks(fd,file_table[free_file].size);
            close(fd);
            }
         }
      else
         fprintf(stderr,"Error: cannot open '%s' for writing\n",
            file_table[free_file].name);
      }
}

/* deletes specified file from disk and file_table */
void delete_file(number)
int number;
{
   if (file_table[number].size)
      {
      if (remove(file_table[number].name))
         fprintf(stderr,"Error: Cannot delete
'%s'\n",file_table[number].name);
      else
         { /* reset entry in file_table and update counter */
         file_table[number].size=0;
         files_deleted++;
         }
      }
}

/* reads entire specified file into temporary buffer */
void read_file(number,buffered)
int number;   /* number of file to read (from file_table) */
int buffered; /* 1=buffered I/O (default), 0=unbuffered I/O */
```

```
{
   FILE *fp=NULL;
   int fd=-1;
   int i;

   if (buffered)
      fp=fopen(file_table[number].name,"r");
   else
      fd=open(file_table[number].name,O_RDONLY,0644);

   if (fp || fd!=-1)
      { /* read as many blocks as possible then read the remainder */
      if (buffered)
         {
         for (i=file_table[number].size; i>=read_block_size; i-
=read_block_size)
            fread(read_buffer,read_block_size,1,fp);


         fread(read_buffer,i,1,fp);

         fclose(fp);
         }
      else
         {
         for (i=file_table[number].size; i>=read_block_size; i-
=read_block_size)
            read(fd,read_buffer,read_block_size);

         read(fd,read_buffer,i);

         close(fd);
         }

      /* increment counters to record transaction */
      bytes_read+=file_table[number].size;
      files_read++;
      }
   else
      fprintf(stderr,"Error: cannot open '%s' for reading\n",
         file_table[number].name);
}

/* appends random data to a chosen file up to the maximum configured
length */
void append_file(number,buffered)
int number;   /* number of file (from file_table) to append date to */
int buffered; /* 1=buffered I/O (default), 0=unbuffered I/O */
{
   FILE *fp=NULL;
   int fd=-1;
   int block; /* size of data to append */

   if (file_table[number].size<file_size_high)
      {
      if (buffered)
         fp=fopen(file_table[number].name,"a");
      else
         fd=open(file_table[number].name,O_RDWR|O_APPEND,0644);

      if ((fp || fd!=-1) && file_table[number].size<file_size_high)
         {
         block=RND(file_size_high-file_table[number].size)+1;

         if (buffered)
            {
            fwrite_blocks(fp,block);
            fclose(fp);
            }
         else
            {
            write_blocks(fd,block);
            close(fd);
            }

         file_table[number].size+=block;
         files_appended++;
         }
      else
         fprintf(stderr,"Error: cannot open '%s' for append\n",
            file_table[number].name);
      }
}

/* finds and returns the offset of a file that is in use from the
file_table */
int find_used_file() /* only called after files are created */
{
   int used_file;

   while (file_table[used_file=RND(simultaneous<<1)].size==0)
      ;

   return(used_file);
}

/* reset global counters - done before each test run */
void reset_counters()
{
   files_created=0;
   files_deleted=0;
   files_read=0;
   files_appended=0;
   bytes_written=0;
   bytes_read=0;
}

/* perform the configured number of file transactions
   - a transaction consisted of either a read or append and either a
     create or delete all chosen at random */
int run_transactions(buffered)
int buffered; /* 1=buffered I/O (default), 0=unbuffered I/O */
{
   int percent; /* one tenth of the specified transactions */
   int i;

   percent=transactions/10;
   for (i=0; i<transactions; i++)
      {
      if (files_created==files_deleted)
         {
         printf("out of files!\n");
         printf("For this workload, either increase the number of files
or\n");
         printf("decrease the number of transactions.\n");
         break;
         }

      if (bias_read!=-1) /* if read/append not locked out... */
         {
         if (RND(10)<bias_read) /* read file */
            read_file(find_used_file(),buffered);
         else /* append file */
            append_file(find_used_file(),buffered);
         }

      if (bias_create!=-1) /* if create/delete not locked out... */
         {
         if (RND(10)<bias_create) /* create file */
            create_file(buffered);
         else /* delete file */
            delete_file(find_used_file());
         }

      if ((i % percent)==0) /* if another tenth of the work is
done...*/
         {
         putchar('.'); /* print progress indicator */
         fflush(stdout);
         }
      }

   return(transactions-i);
}

char **build_location_index(list,weight)
file_system *list;
int weight;
{
   char **index;
   int count;
   int i=0;

   if ((index=(char **)calloc(1,weight*sizeof(char *)))==NULL)
      fprintf(stderr,"Error: cannot build weighted index of
locations\n");
   else
      for (; list; list=list->next)
         for (count=0; count<list->system.size; count++)
            index[i++]=list->system.name;
   return(index);
}

void create_subdirectories(dir_list,base_dir,subdirs)
file_system *dir_list;
char *base_dir;
int subdirs;
{
   char dir_name[MAX_LINE+1]; /* buffer holding subdirectory names */
   char save_dir[MAX_LINE+1];
   int i;

   if (dir_list)
      {
      for (; dir_list; dir_list=dir_list->next)
         create_subdirectories(NULL,dir_list->system.name,subdirs);
      }
   else
      {
      if (base_dir)
         sprintf(save_dir,"%s%s",base_dir,SEPARATOR);
      else
         *save_dir='\0';

      for (i=0; i<subdirs; i++)
         {
         sprintf(dir_name,"%ss%d",save_dir,i);
         MKDIR(dir_name);
         }
      }
}

void delete_subdirectories(dir_list,base_dir,subdirs)
file_system *dir_list;
char *base_dir;
int subdirs;
{
   char dir_name[MAX_LINE+1]; /* buffer holding subdirectory names */
   char save_dir[MAX_LINE+1];
   int i;

   if (dir_list)
      {
      for (; dir_list; dir_list=dir_list->next)
         delete_subdirectories(NULL,dir_list->system.name,subdirs);
      }
   else
      {
      if (base_dir)
         sprintf(save_dir,"%s%s",base_dir,SEPARATOR);
      else
         *save_dir='\0';

      for (i=0; i<subdirs; i++)
         {
         sprintf(dir_name,"%ss%d",save_dir,i);
         rmdir(dir_name);
         }
      }
}

/* CLI callback for 'run' - benchmark execution loop */
int cli_run(param) /* none */
char *param; /* unused */
{
   time_t start_time,t_start_time,t_end_time,end_time; /* elapsed
timers */
   int delete_base; /* snapshot of deleted files counter */
   FILE *fp=NULL; /* file descriptor for directing output */
   int incomplete;
   int i; /* generic iterator */

   reset_counters(); /* reset counters before each run */

   sgenrand(seed); /* initialize random number generator */

   /* allocate file space and fill with junk */
   file_source=initialize_file_source(file_size_high<<1);

   /* allocate read buffer */
   read_buffer=(char *)malloc(read_block_size);

   /* allocate table of files at 2 x simultaneous files */
   file_allocated=0;
   if ((file_table=(file_entry
*)calloc(simultaneous<<1,sizeof(file_entry)))==
      NULL)
      fprintf(stderr,"Error: Failed to allocate table for %d files\n",
         simultaneous<<1);

   if (file_system_count>0)

location_index=build_location_index(file_systems,file_system_weight);

   /* create subdirectories if necessary */
   if (subdirectories>1)
      {
      printf("Creating subdirectories...");
      fflush(stdout);
      create_subdirectories(file_systems,NULL,subdirectories);
      printf("Done\n");
      }

   time(&start_time); /* store start time */

   /* create files in specified directory until simultaneous number */
   printf("Creating files...");
   fflush(stdout);
   for (i=0; i<simultaneous; i++)
      create_file(buffered_io);
   printf("Done\n");

   printf("Performing transactions");
   fflush(stdout);
   time(&t_start_time);
   incomplete=run_transactions(buffered_io);
   time(&t_end_time);
   if (!incomplete)
      printf("Done\n");

   /* delete remaining files */
   printf("Deleting files...");
   fflush(stdout);
   delete_base=files_deleted;
   for (i=0; i<simultaneous<<1; i++)
      delete_file(i);
   printf("Done\n");

   /* print end time and difference, transaction numbers */
   time(&end_time);

   /* delete previously created subdirectories */
   if (subdirectories>1)
      {
      printf("Deleting subdirectories...");
      fflush(stdout);
      delete_subdirectories(file_systems,NULL,subdirectories);
      printf("Done\n");
      }

   if (location_index)
      {
      free(location_index);
      location_index=NULL;
      }

   if (param)
      if ((fp=fopen(param,"a"))==NULL)
         fprintf(stderr,"Error: Cannot direct output to file
'%s'\n",param);

   if (!fp)
      fp=stdout;

   if (!incomplete)
      reports[report](fp,end_time,start_time,t_end_time,t_start_time,
         files_deleted-delete_base);

   if (param && fp!=stdout)
      fclose(fp);

   /* free resources allocated for this run */
   free(file_table);
   free(read_buffer);
   free(file_source);

   return(1); /* return 1 unless exit requested, then return 0 */
}

/* CLI callback for 'show' - print values of configuration variables */
int cli_show(param)
```

```
char *param; /* optional: name of output file */
{
   char current_dir[MAX_LINE+1]; /* buffer containing working directory
*/
   file_system *traverse;
   FILE *fp=NULL; /* file descriptor for directing output */

   if (param)
      if ((fp=fopen(param,"a"))==NULL)
         fprintf(stderr,"Error: Cannot direct output to file
'%s'\n",param);

   if (!fp)
      fp=stdout;

   fprintf(fp,"Current configuration is:\n");
   fprintf(fp,"The base number of files is %d\n",simultaneous);
   fprintf(fp,"Transactions: %d\n",transactions);

   if (file_size_low!=file_size_high)
      {
      fprintf(fp,"Files range between %s ",scale(file_size_low));
      fprintf(fp,"and %s in size\n",scale(file_size_high));
      }
   else
      fprintf(fp,"Files are %s in size\n",scale(file_size_low));

   fprintf(fp,"Working director%s:
%s\n",(file_system_count>1)?"ies":"y",
      (file_system_count==0)?GETWD(current_dir):"");

   for (traverse=file_systems; traverse; traverse=traverse->next)
      printf("\t%s (weight=%d)\n",traverse->system.name,traverse-
>system.size);

   if (subdirectories>0)
      fprintf(fp,"%d subdirector%s will be used\n",subdirectories,
         (subdirectories==1)?"y":"ies");

   fprintf(fp,"Block sizes are: read=%s, ",scale(read_block_size));
   fprintf(fp,"write=%s\n",scale(write_block_size));
   fprintf(fp,"Biases are: read/append=%d,
create/delete=%d\n",bias_read,
      bias_create);
   fprintf(fp,"%ssing Unix buffered file I/O\n",buffered_io?"U":"Not
u");
   fprintf(fp,"Random number generator seed is %d\n",seed);

   fprintf(fp,"Report format is %s.\n",report?"terse":"verbose");

   if (param && fp!=stdout)
      fclose(fp);

   return(1); /* return 1 unless exit requested, then return 0 */
}

/* CLI callback for 'quit' - returns 0 causing UI to exit */
```

```
int cli_quit(param) /* none */
char *param; /* unused */
{
   return(0); /* return 1 unless exit requested, then return 0 */
}

/* CLI callback for 'help' - prints help strings from command_list */
int cli_help(param)
char *param; /* optional: specific command to get help for */
{
   int n=0; /* number of matching items */
   int i; /* traversal variable for command table */
   int len;

   if (param && (len=strlen(param))>0) /* if a command is specified...
*/
      for (i=0; command_list[i].name; i++) /* walk command table */
         if (!strncmp(command_list[i].name,param,len))
            {
            printf("%s -
%s\n",command_list[i].name,command_list[i].help);
            n++;
            }

   if (!param || !n)
      for (i=0; command_list[i].name; i++) /* traverse command table */
         printf("%s - %s\n",command_list[i].name,command_list[i].help);

   return(1); /* return 1 unless exit requested, then return 0 */
}

/* read CLI line from user, translate aliases if any, return fgets
status */
char *cli_read_line(buffer,size)
char *buffer; /* empty input line */
int size;
{
   char *result;

   printf("%s",PROMPT);                 /* print prompt */
   fflush(stdout);                      /* force prompt to print */
   if (result=fgets(buffer,size,stdin)) /* read line safely */
      {
      buffer[strlen(buffer)-1]='\0';    /* delete final CR */
      if (!strcmp(buffer,"?"))           /* translate aliases */
         strcpy(buffer,"help");
      if (!strcmp(buffer,"exit"))
         strcpy(buffer,"quit");
      }

   return(result);                      /* return success of fgets */
}

/* parse CLI input line */
int cli_parse_line(buffer)
char *buffer; /* line of user input */
{
   int result=1; /* default return status */
   int len; /* length of parsed command */
   int i; /* traversal variable for command table */

   if (*buffer=='!') /* check for shell escape */
      system((strlen(buffer)>1)?buffer+1:getenv("SHELL"));
   else
      {
      for (i=0; command_list[i].name; i++) /* walk command table */
         if (!strncmp(command_list[i].name,buffer,
            len=strlen(command_list[i].name)))
            { /* if command matches... */
            result=(command_list[i].func)
               (((int)strlen(buffer)>len)?buffer+len+1:NULL);
            break; /* call function and pass remainder of line as
parameter */
            }

      if (!command_list[i].name) /* if no commands were called... */
         printf("Eh?\n"); /* tribute to Canadian diction */
      }

   return(result); /* return 1 unless exit requested, then return 0 */
}

/* read config file if present and process it line by line
   - if 'quit' is in file then function returns 0 */
int read_config_file(filename,buffer)
char *filename; /* file name of config file */
char *buffer;   /* temp storage for each line read from file */
{
   int result=1; /* default exit value - proceed with UI */
   FILE *fp;

   if (fp=fopen(filename,"r")) /* open config file */
      {
      printf("Reading configuration from file '%s'\n",filename);
      while (fgets(buffer,MAX_LINE,fp) && result) /* read lines until
'quit' */
         {
         buffer[strlen(buffer)-1]='\0'; /* delete final CR */
         result=cli_parse_line(buffer); /* process line as typed in */
         }

      fclose(fp);
      }

   return(result);
}

/* main function - reads config files then enters get line/parse line
loop */
main(argc,argv)
int argc;
char *argv[];
{
   char buffer[MAX_LINE+1]; /* storage for input command line */
   printf("PostMark v1.5 : 3/27/01\n");
   if (read_config_file((argc==2)?argv[1]:".pmrc",buffer))
      while (cli_read_line(buffer,MAX_LINE) && cli_parse_line(buffer))
         ;
}

/*

                         The "Artistic License"

                                Preamble

The intent of this document is to state the conditions under which a
Package may be copied, such that the Copyright Holder maintains some
semblance of artistic control over the development of the package,
while giving the users of the package the right to use and distribute
the Package in a more-or-less customary fashion, plus the right to make
reasonable modifications.

Definitions:

        "Package" refers to the collection of files distributed by the
        Copyright Holder, and derivatives of that collection of files
        created through textual modification.

        "Standard Version" refers to such a Package if it has not been
        modified, or has been modified in accordance with the wishes
        of the Copyright Holder as specified below.

        "Copyright Holder" is whoever is named in the copyright or
        copyrights for the package.

        "You" is you, if you're thinking about copying or distributing
        this Package.

        "Reasonable copying fee" is whatever you can justify on the
        basis of media cost, duplication charges, time of people
involved,
        and so on.  (You will not be required to justify it to the
        Copyright Holder, but only to the computing community at large
        as a market that must bear the fee.)

        "Freely Available" means that no fee is charged for the item
        itself, though there may be fees involved in handling the item.
        It also means that recipients of the item may redistribute it
        under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the
Standard Version of this Package without restriction, provided that you
duplicate all of the original copyright notices and associated
disclaimers.

2. You may apply bug fixes, portability fixes and other modifications
derived from the Public Domain or from the Copyright Holder.  A Package
modified in such a way shall still be considered the Standard Version.
```

```
/* A C-program for MT19937: Integer version (1999/10/28)          */
/*  genrand() generates one pseudorandom unsigned integer (32bit) */
/* which is uniformly distributed among 0 to 2^32-1  for each     */
/* call. sgenrand(seed) sets initial values to the working area   */
/* of 624 words. Before genrand(), sgenrand(seed) must be         */
/* called once. (seed is any 32-bit integer.)                     */
/*   Coded by Takuji Nishimura, considering the suggestions by    */
/* Topher Cooper and Marc Rieffel in July-Aug. 1997.              */

/* This library is free software; you can redistribute it and/or   */
/* modify it under the terms of the GNU Library General Public     */
/* License as published by the Free Software Foundation; either    */
/* version 2 of the License, or (at your option) any later         */
/* version.                                                        */
/* This library is distributed in the hope that it will be useful, */
/* but WITHOUT ANY WARRANTY; without even the implied warranty of  */
/* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.            */
/* See the GNU Library General Public License for more details.    */
/* You should have received a copy of the GNU Library General      */
/* Public License along with this library; if not, write to the    */
/* Free Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA   */
/* 02111-1307  USA                                                 */

/* Copyright (C) 1997, 1999 Makoto Matsumoto and Takuji Nishimura. */
/* Any feedback is very welcome. For any question, comments,       */
/* see http://www.math.keio.ac.jp/matumoto/emt.html or email       */
/* matumoto@math.keio.ac.jp                                        */

/* REFERENCE                                                       */
/* M. Matsumoto and T. Nishimura,                                  */
/* "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform  */
/* Pseudo-Random Number Generator",                                */
/* ACM Transactions on Modeling and Computer Simulation,           */
/* Vol. 8, No. 1, January 1998, pp 3--30.                          */

/* Period parameters */
#define N 624
#define M 397
#define MATRIX_A 0x9908b0df   /* constant vector a */
#define UPPER_MASK 0x80000000 /* most significant w-r bits */
#define LOWER_MASK 0x7fffffff /* least significant r bits */

/* Tempering parameters */
#define TEMPERING_MASK_B 0x9d2c5680
#define TEMPERING_MASK_C 0xefc60000
#define TEMPERING_SHIFT_U(y)  (y >> 11)
#define TEMPERING_SHIFT_S(y)  (y << 7)
#define TEMPERING_SHIFT_T(y)  (y << 15)
#define TEMPERING_SHIFT_L(y)  (y >> 18)

static unsigned long mt[N]; /* the array for the state vector  */
static int mti=N+1; /* mti==N+1 means mt[N] is not initialized */

/* Initializing the array with a seed */
void
sgenrand(seed)
    unsigned long seed;
{
    int i;

    for (i=0;i<N;i++) {
         mt[i] = seed & 0xffff0000;
         seed = 69069 * seed + 1;
         mt[i] |= (seed & 0xffff0000) >> 16;
         seed = 69069 * seed + 1;
    }
    mti = N;
}

/* Initialization by "sgenrand()" is an example. Theoretically,      */
/* there are 2^19937-1 possible states as an intial state.           */
/* This function allows to choose any of 2^19937-1 ones.             */
/* Essential bits in "seed_array[]" is following 19937 bits:         */
/*  (seed_array[0]&UPPER_MASK), seed_array[1], ..., seed_array[N-1]. */
/* (seed_array[0]&LOWER_MASK) is discarded.                           */
/* Theoretically,                                                     */
/*  (seed_array[0]&UPPER_MASK), seed_array[1], ..., seed_array[N-1]  */
/* can take any values except all zeros.                              */
void
lsgenrand(seed_array)
    unsigned long seed_array[];
    /* the length of seed_array[] must be at least N */
{
    int i;

    for (i=0;i<N;i++)
      mt[i] = seed_array[i];
    mti=N;
}

unsigned long
genrand()
{
    unsigned long y;
    static unsigned long mag01[2]={0x0, MATRIX_A};
    /* mag01[x] = x * MATRIX_A  for x=0,1 */

    if (mti >= N) { /* generate N words at one time */
        int kk;

        if (mti == N+1)   /* if sgenrand() has not been called, */
            sgenrand(4357); /* a default initial seed is used   */

        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1];

        mti = 0;
    }

    y = mt[mti++];
    y ^= TEMPERING_SHIFT_U(y);
    y ^= TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B;
    y ^= TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C;
    y ^= TEMPERING_SHIFT_L(y);

    return y;
}
```

Appendix D - OSCP Testing Procedure

Tests Environment

The Oracle Storage system certification test SDK comes typically with a tar file. Various directories are created after one untars the tar file. For the fault injection tests, you must place $ORALE_HOME/dbs, $ORALCE_HOME/rdbms/log on the network storage system. Almost all the tests below places the database datafiles, logfiles, and control files on the network storage system. Only one test requires placing some database datafiles on the local disk of the NFS client where Oracle instance runs.

Most tests described below follow a 3 step scenario: (1) start an intensive database workload by running test tknfsutil, (2) cause a physical failure by a person, such as powering off the NFS server, (3) validate that the database survived the failure.

To start the intensive database workload, issue command "sdk_runqa nfs scenario-1". This workload includes changing some database tables while maintaining some invariant. This test runs by self for 2 hours. You may terminate the test any time by simply shutdown abort the instance.

You may validate the integrity of the database either after the test has been terminated, or while the test is running.

1. startup database if it is not up already
2. SVRMGR> connect sys/knl_test7 as sysdba
3. SVRMGR> set SERVEROUTPUT ON
4. SVRMGR> execute checkIntegrity();

The PL/SQL routine checkIntegrity() outputs the following to the screen:

```
Integrity check succeeded

Most recent update: Mar 12, 1999, 5:60:45
```

If the invariant is no longer true, which means he database is no longer consistent, checkIntegrity() outputs the following to the screen:

```
Integrity check failed

Most recent update: Mar 12, 1999, 5:60:45
```

OSCP Tests

Synchronous writes

This test assumes that Oracle runs on a machine separate from the network file server. It requires some database data files to be placed on the local disk of the NFS client where Oracle runs. The integrity of Oracle database relies on the fact that Oracle follows the "log ahead" protocol – the redo for a change must be on persistent media first before the change itself can be on persistent media. Thus, if write is not synchronous, the following test may fail. The steps are as follows:

1. Make sure no Oracle instance is up. Use NFS hard mount.

2. Run test tknfsutil2 by issuing command "sdk_runqa nfs scenario-2 dbs_dir-/your/dbs/directory"

3. Power off the network file server.

4. Shutdown Oracle via shutdown abort.

5. Restart the network file server, and restart Oracle. Make sure Oracle can complete crash recovery and restart successfully. If not, this test fails.

6. Run checkIntegrity(). If it succeeds, this test passes. Otherwise this test fails.

The test above should be repeated at least 5 times. One should only consider that this test passed when all the runs succeed.

Hard-mount/Soft-mount

Either NFS hard-mount or soft-mount works with Oracle. The two different modes cause different behavior. If hard mount is used, Oracle should stay up while NFS server is down; it simply stalls. Because the I/O made by Oracle do not have timeout, Oracle should be able to continue from where it stalled once NFS is working again. If soft mount is used, Oracle may crash because of I/O error if the network file is down.

To test hard mount, perform the following steps:

1. Start the work load by running test tknfsutil.

2. Manually cause faults to bring NFS down. This can be done by power-off the server, or disconnect the network cable.

3. Restore NFS by either rebooting the server or connect the network cable.

4. Make sure Oracle does not crash, and it can continue once NFS is restored.

5. Start a session, connect as “sys/knl_test7 as sysdba”, run checkIntegrity(). The check must succeed. Observe the most recent updated time printed by chckIntegrity(). This time should be after the NFS has been restored.

To test soft mount, perform the following:

1. Start the work load by running test tknnfsutil.
2. Manually cause faults to bring down NFS. This can be done by power-off the server, or disconnect network cable. Oracle should crash after seeing I/O error.
3. Restart Oracle instance. Make sure Oracle can complete crash recovery and restart successfully.
4. Run checkIntegrity(). It must succeed.

<u>NVRAM</u>

If the NVRAM for the filer fails, the user in general must assume the whole network file server has been damaged, and must restore a backup of all the database files on the network file server, and perform database media recovery.

There are two ways to get around this problem.

(1) mirror NVRAMs

(2) Guarantee that the filer will appear to be at a consistent state at some point in the past.

In this document, we will call the latter mechanism the “snapshot” mechanism.

If the snapshot mechanism is implemented, the user may avoid restoring all the files on the filer. However, one must put Oracle control files, log files, and all datafiles on the same filer. The log archived before the NVRAM failure must not be mixed with the logs archived after the NVRAM failure. In general, one should take a full database backup after the NVRAM failure, and discard the logs archived before the NVRAM failure. Note that, in this approach, some committed transactions may be lost in case NVRAM fails.

The issues associated with NVRAMs are as follows:

When all NVRAMs fail, the storage system must somehow bring down the Oracle instance, regardless of whether the “snapshot” mechanism is implemented. This is necessary to make sure (1) stale data is not returned to Oracle, and (2) Oracle’s buffer cache will not mix current database blocks with previous storage system “snapshots”. The storage system can bring down the Oracle instance in two ways: (1) the storage system may return I/O errors to all I/O requests directed to the filer immediately after the NVRAM fails. These I/O error will bring down the Oracle instance. (2) the storage system may simply stops returning any I/O requests after NVRAM fails, causing Oracle to either hang (hard mount) or abort after I/O timeout (soft mount). In the second case, a database administrator must shutdown abort the hanging instance.

The storage system must somehow inform the user (database administrator) that all NVRAMs have failed, regardless of whether “snapshot” mechanism is implemented. This is necessary so that, in case the snapshot mechanism is not implemented, the database administrator knows that he or she must restore all database file on the network storage system and perform media recovery. In case snapshot mechanism is implemented, the database administrator must also be informed because some committed transactions may be lost as a result of going to previous “snapshots”.

When installing a new NVRAM, there should be a way to clear the contents of the new NVRAM, so that the new NVRAM will not bring old unrelated data to the storage system.

NVRAM Test 1: Make sure NVRAM failure bring down Oracle instance

This test assumes that, after all NVRAM fails, all I/O are expected to return I/O errors.

1. Make sure Oracle instance is not up. Test both hard mount and soft mount.
2. Start Oracle instance
3. Start a workload by running test tknfsutil1.
4. Cause all NVRAM to fail.
5. Make sure that Oracle instance is down immediately after the NVRAM failure because of I/O errors.

NVRAM Test 2: Make sure NVRAM failure causes Oracle instance to hang

This test assumes that, after all NVRAM fails, the network storage system will stop responding to any I/O requests.

1. Make sure Oracle instance is not up. Use hard mount.
2. Start Oracle.
3. Start a workload by running test tknfsutil2.

4. Start a server manager session, connect as “sys/knl_test7 as sysdba”

5. Cause all NVRAM to fail.

6. In the user session, type the following command:

    Select * from nfs_vnram_failure;

7. Make sure no data is returned. You may then shutdown abort the instance. Make sure that succeeds

Table nfs_nvram_failure is not manipulated in test tknfsutil2. Thus its data can not possibly in the buffer cache of the Oracle instance. If the select returned any data, it must come from the disk of the network storage system. These data can potentially be false.

NVRAM Test 3: Test the “snapshot” mechanism

This test checks if the “snapshot” mechanism can bring the database to a previous consistent state.

1. Perform either NVRAM test 1 or NVRAM test 2. Make sure NVRAM failure either brings down the instance or stops the instance.

2. Make sure the instance is not up.

3. Enable network storage system to go to a previous “snapshot”.

4. Start up the instance. Connect as user “sys/knl_test7 as sysdba”

5. Run checkIntegrity(). This test fails if checkIntegrity() outputs “fail”.

6. Compare the most recent update time printed out ny checkIntegrity() with the current system time. Make sure the difference is within a reasonable limit. (All transactions after the most recent update time is lost.)

NVRAM Test 4: Installing new NVRAMs

1. Perform either NVRAM test 1 r NVRAM test 2. Make sure NVRAM failure either brings down the instance or stops the instance.

2. Make sure the instance is not up.

3. Install a different NVRAM.

4. Start up the instance. Connect as user “sys/knl_test7 as sysdba”

5. Run checkIntegrity(). Make sure it succeeds.

File Locks

The steps of this test are as follows:

1. Make sure no Oracle instance is up. Use NFS hard mount.

2. Run test tknfsutil by issuing command “sdk_runqa nfs scenario=3”

3. Power off the filer. One should see that the Oracle instance simply stalls waiting for NFS to respond.

4. Power of the filer. One should see that the test continues to run.

5. Attempt to start a second Oracle instance by issuing command “sdk_runqa nfs scenario=3”. This attempt should fail.

Oracle uses file locks to prevent user from starting a second instance in an non-OPS (Oracle Parallel Server) configuration. If file locks are not preserved across network file server failures, the above test may fail.

Double Failures

Double Failure Test 1: Two Server failures

1. Start the work load by running test tknfsutil1. Use hard mount.

2. Power off the filer

3. Power on the filer

4. While the filer is rebooting and contacting NFS clients, power off the filer again.

5. Power on the filer and make sure Oracle continues without errors.

6. Start a session. Connect as “sys/knl_test7 as sysdba”. Run checkIntegrity(). Make sure it succeeds.

Repeat this test for at least 3 times.

Double Failure Test 2: Server failure followed by a client failure while the server is rebooting

1. Start the work load by running test tknfsutil1. Use hard mount.
2. Power off the filer
3. Power on the filer
4. While the filer is rebooting and contacting NFS clients, poer off the NFS client machine on which Oracle runs.
5. Reboot the machine and restart Oracle. Make sure Oracle can complete crash recovery and restart successfully.
6. Connect as “sys/knl_test7 as sysdba”. Run checkIntegrity(). Make sure it succeeds.

Repeat this test for at least 3 times.

<u>Stale Cache</u>

A NFS implementation may have a write-through cache on a NFS client. When Oracle is shut down on the client, the network file server must make sure that Oracle does not fetch stale cache blocks when it restarts on the same node.

To test this, set up two machines that are connected to the filer.

1. Start Oracle on one machine by running the command “sdk_runqa nfs scenario=4 machine=1”.
2. Start Oracle on the second machine by running the command “sdk_runqa nfs scenario=4 machine=2”.
3. Restart Oracle on the first machine. Make sure you see the changes made on the second machine
4. Run the following command from the second machine:

```
Select * from nfs_stale_cache;
```

If you see the value “machine 2” then your test has passed, otherwise you are seeing value from a stale cache.

Appendix E – MC/ServiceGuard for Linux

1. Getting Started with MC/ServiceGuard for Linux - http://docs.hp.com/linux/pdf/B9903-90006.pdf
2. Managing MC/ServiceGuard for Linux - http://docs.hp.com/linux/pdf/B9903-90005.pdf

Appendix F - HP DS 2300

DS 2300 Features:

- Disk array type – JOBD
- Interconnect – Ultra-160
- Max disks – 14
- Ports – Two (2) 68-ping VHDCI ports per bus-controller card
- Redundant power supply
- Supports – MC/Service Guard, MSCS (Microsoft Cluster Services)

Source:
http://www.hp.com/products1/storage/products/disk_arrays/disksystems/ds2300/index.html

Appendix G – MC/ServiceGuard for Linux

Managing MC/ServiceGuard NFS for Linux - http://docs.hp.com/linux/pdf/T1442-90002.pdf

Appendix H – Oracle 9i Real Application Clusters References

1. Oracle 9i RAC on HP-UX - http://otn.oracle.com/products/oracle9i/htdocs/9iRAChp.html#54
2. Oracle 9i RAC Cache Fusion Delivers Scalability, An Oracle White Paper by Sohan DeMel, February 2002 - http://otn.oracle.com/products/oracle9i/pdf/cache_fusion_rel2.pdf
3. Oracle 9i DBA Guide Release 2 (9.2) - http://docs.oracle.com/cd_a97630/server.920/a96521/create.htm#998107

Appendix I

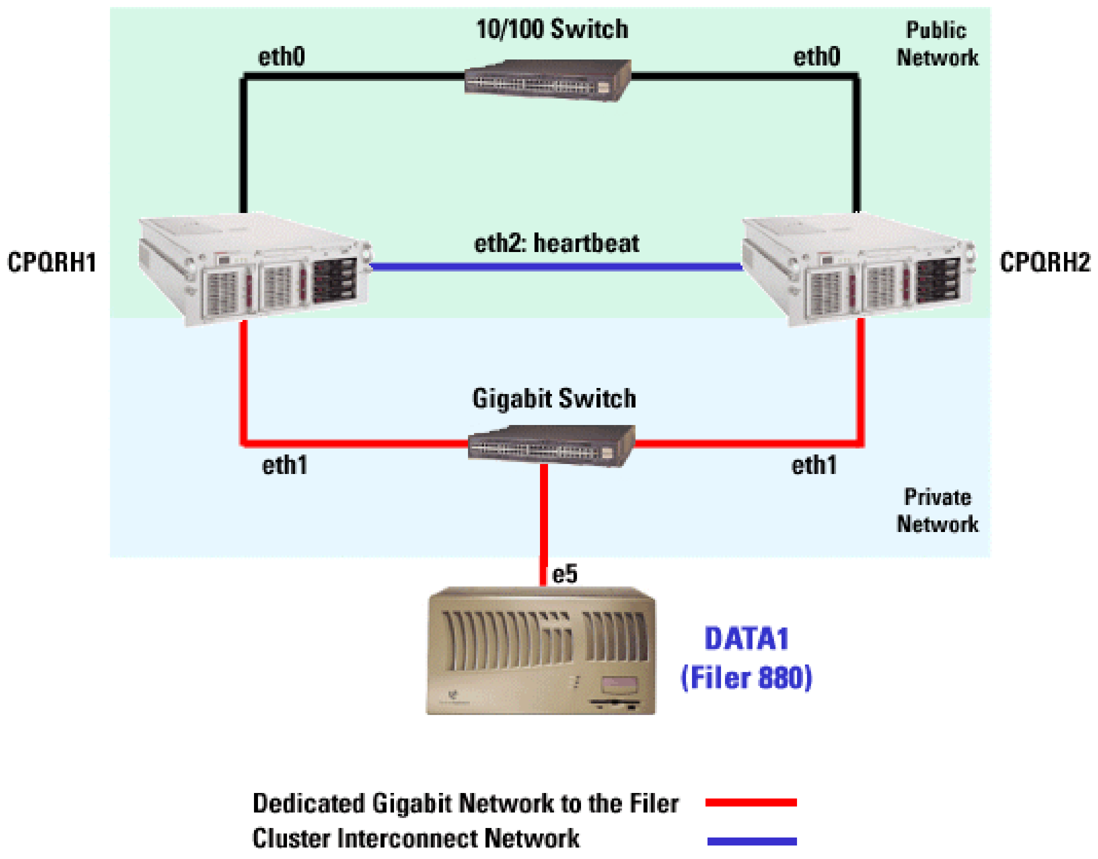
1. Oracle Parallel Server and Windows 2000 Advanced Server on IBM Netfinity - http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245449.pdf
2. Oracle Parallel Processing By Tushar Mahapatra, Sanjay Mishra, August 2000 ISBN 1-56592-701-X

Appendix J

Network Appliances Certified ORAC Configuration 1

Oracle 9i RAC on Compaq Proliant Servers Running Linux 7 and F880

The configuration presented here is based on the Oracle9*i* RAC certification environment specified by Oracle and Network Appliance.



The above figure illustrates a typical configuration of Oracle9*i* RAC with a Netapp filer. The Intel boxes used in this configuration are from Compaq (model DL580R01). This is a scalable configuration and allows the user to scale horizontally and internally in terms of processor, memory, and storage.

As shown in the network diagram, we recommend that you dedicate a private network connection between the Oracle9*i* RAC servers and the filer. This is accomplished using a dedicated Gigabit network (with a Gigabit switch) to the filer. A dedicated network connection is beneficial for the following reasons:

- In a 9*i* RAC environment it is important to eliminate any contentions and latencies.
- By providing a separate network, security is ensured.

The cluster interconnect is used to monitor the heartbeat of the two servers in the cluster. This is a typical configuration that can be deployed in a customer's environment.

Cluster Components

Cluster Nodes

- 2 * 2-way 900 MHz, 2GB RAM Compaq Proliant DL580 Servers
- 2 * Intel III Gigabit Ethernet NICs (for cluster interconnect)
- 2 * Intel III Gigabit Ethernet NICs (for Network Appliance filer I/O)

Storage Infrastructure

- 1 * Network Appliance 880 filer with Data ONTAP™ 6.1.2R3
- 1 * 4/8 port Gigabit switch
- 1 * Gigabit NIC in the filer
- 1 or more disk shelves based on the disk space requirements

Software Requirements
(For both the nodes in the participating cluster unless specified otherwise)

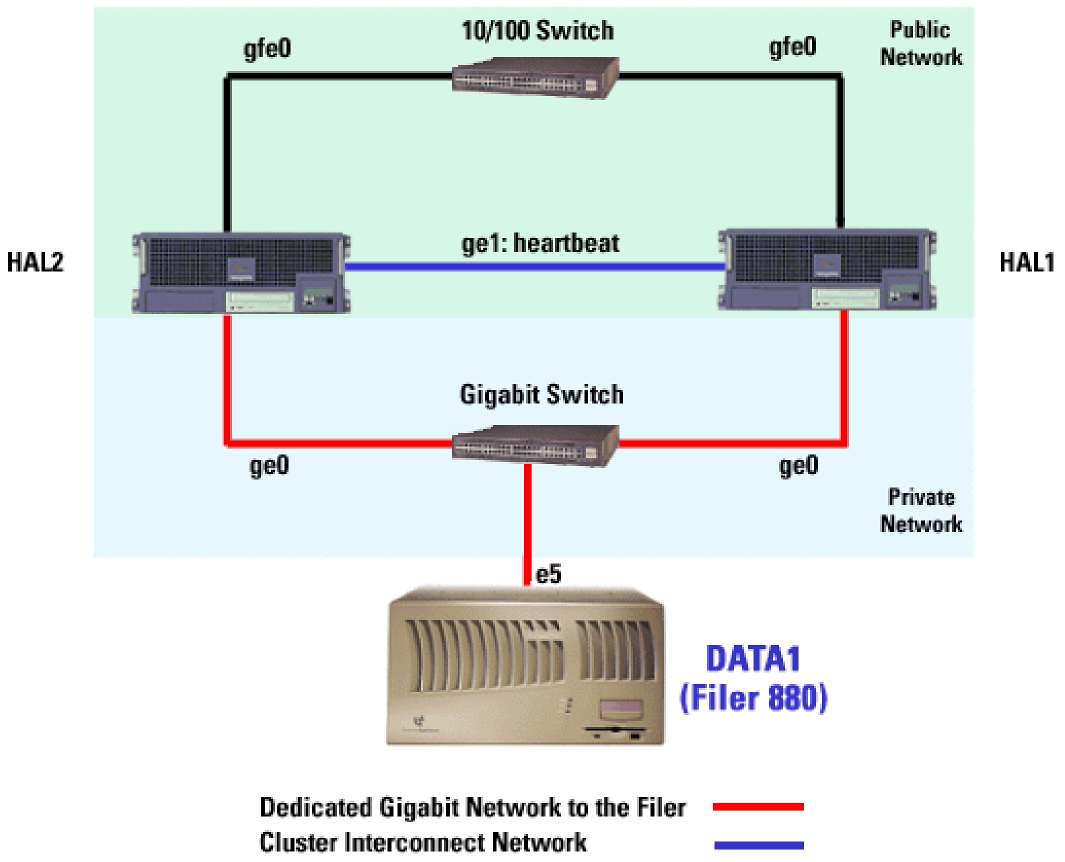
- Red Hat Linux 7.1, kernel 2.4.9-31
- Oracle9*i*, release 9.0.1 with Real Application Cluster license
- Oracle9*i* - 9.0.1.3 patch set
- Intel III Gigabit Ethernet driver for the Gigabit NICs

Source: Network Appliances - http://www.netapp.com/tech_library/3164.html

Network Appliances Certified ORAC Configuration 2

Oracle 9i RAC on Fujitsu Primepower Servers running Solaris 8 and F880

The configuration presented here is based on the Oracle9*i* RAC certification environment specified by Oracle and Network Appliance.



The above figure illustrates a typical configuration of Oracle9*i* RAC with a Netapp filer and Fujitsu-Siemens computers. This is a scalable configuration and allows the user to scale horizontally and internally in terms of processor, memory, and storage.

As shown in the network diagram, we recommend that you dedicate a private network connection between the Oracle9*i* RAC servers and the filer. This is accomplished using a dedicated Gigabit network (with a Gigabit switch) to the filer. A dedicated network connection is beneficial for the following reasons:

- In a 9*i* RAC environment it is important to eliminate any contentions and latencies.
- By providing a separate network, security is ensured.

The cluster interconnect is used to monitor the heartbeat of the two servers in the cluster. This is a typical configuration that can be deployed in a customer's environment.

Cluster Components

Cluster Nodes

- 2 * Fujitsu Primepower Servers
- 2 * Sun Gigabit Ethernet NICs (for cluster interconnect)
- 2 * Sun Gigabit Ethernet NICs (for Network Appliance filer I/O)

Storage Infrastructure

- 1 * Network Appliance 8XX filer with Data ONTAP™ 6.1.R1
- 1 * 4/8 port Gigabit switch
- 1 * Gigabit NIC in the filer
- 1 or more disk shelves based on the disk space requirements

Software Requirements
(For both the nodes in the participating cluster unless specified otherwise.)

- Solaris 8 with the latest version of kernel patches described below:
    - 108813-08 SunOS 5.8: Sun Gigabit Ethernet 3.0
    - 108806-08 SunOS 5.8: Sun Quad FastEthernet qfe driver
    - 108528-13 SunOS 5.8: kernel update patch
    - 108727-14 SunOS 5.8: /kernel/fs/nfs and /kernel/fs/sparcv9/nfs patch
- PRIMECLUSTER 4.0 or higher
    - CF V4.0A
    - PAS V4.0A
    - RMS V4.0A (required logically for node elimination; no interface dependency)
- Oracle 9*i*, release 9.0.1 with Real Application Cluster license
- Fujitsu PRIMECLUSTER RAC package for Oracle 9.0.1
- Sun Gigabit Ethernet-3.0 driver for the Gigabit NICs

Source: Network Appliances - http://www.netapp.com/tech_library/3165.html

References

1. Oracle9i for UNIX: HP/UX 11.0 and 11i Best Practices with a NetApp® Filer - http://www.netapp.com/tech_library/3146.html#1.
2. Oracle7 for UNIX: Backup and Recovery Using a NetApp Filer - http://www.netapp.com/tech_library/3036.html
3. Oracle8*i*/Oracle8 for UNIX: Backup and Recovery Using a NetApp Filer - http://www.netapp.com/tech_library/3049.html
4. Oracle9*i* for UNIX: Backup and Recovery Using a NetApp Filer - http://www.netapp.com/tech_library/3130.html
5. Oracle8/Oracle8i for UNIX: Integrating with a Network Appliance Filer - http://www.netapp.com/tech_library/3047.html#3.2.
6. Architecting a Gigabit SAN for Database Application Environments - http://www.netapp.com/tech_library/3112.html#2.
7. Oracle8 for UNIX: Integrating with a NetApp Cluster - http://www.netapp.com/tech_library/3046.html
8. Oracle9i for UNIX: Integrating with a Network Appliance Filer - http://www.netapp.com/tech_library/3129.html
9. Oracle8 for UNIX: Providing Disaster Recovery with NetApp SnapMirror Technology - http://www.netapp.com/tech_library/3057.html#4.1.
10. Cisco Systems and Network Appliance:
    Network-Attached Storage Solutions - http://www.netapp.com/tech_library/3121.html#3
11. Using PCI-1000-BaseT and HSC/PCI-1000-Base SX (Gigabit Ethernet) - http://docs.hp.com/hpux/pdf/J1643-90010.pdf
12. Managing MC/ServiceGuard NFS for HP 9000 - http://docs.hp.com/hpux/pdf/B5140-90011.pdf
13. NFS Performance tuning for HPUX 11/11i - http://docs.hp.com/hpux/onlinedocs/netcom/NFS_perf_tuning_hpux110_11i.pdf
14. PCI Gigabit Ethernet Performance - http://docs.hp.com/hpux/onlinedocs/netcom/gbe_perf_final.pdf
15. Boosting Server-to-server Gigabit Throughout with Jumbo Frames - http://docs.hp.com/hpux/onlinedocs/netcom/jumbo_final.pdf
16. INPUT Study: NAS Delivers Lower TCO - http://www.netapp.com/partners/oracle/NAS_DeliversLowerTCOthanSAN.pdf
17. INPUT Study: Database Storage Solutions – A Competitive Study of TCO - http://www.netapp.com/tech_library/ftp/3102.pdf
18. NAD Market Trend by Chris Conner - http://www.garpinvestor.com/nas.htm
19. EMC Leads in NAS Revenue Share - http://www.emc.com/news/press_releases/view.jsp?id=1266
20. Industry report confirms EMC leads in NAS revenue share - http://biz.yahoo.com/bw/020514/142288_3.html
21. Sun joins NAS market - http://www.itworld.com/Comp/3761/ITW924/
22. Gartner Dataquest Report on storage market 2001 - http://www.enterprisestorageforum.com/industrynews/article/0,,1291_1166681,00.html
23. NAS market poised for dynamic growth by Dataquest, May 2000 - http://www.businesssolutionsmag.com/Articles/2000_05_15/00051502.htm
24. NAS – Procom 1999 Annual Report - http://www.procom.com/company/investor/reports/Procom99AR.pdf

25. NAS market 2 billion in 2000 and expected to exceed 10 billion in 2004 - http://www.instat.com/pr/2000/ln0005ms_pr.htm
26. Oracle Usage Guide for Sun StorEdge N8600 Filer - http://www.sun.com.au/products/storage/hardware/nas/pdf/n8600_oracle_usage_guide.pdf
27. Compaq Storage White Papers - http://www.compaq.com/storage/whitepapers.html
28. Trade Report: SAN+NAS, April 2002 - http://www.asia.cnet.com/itmanager/tech/0,39006407,39027064,00.htm
29. SAN+NAS Winter Group Report for EMC - http://www.emc.com/pdf/products/celerra_file_server/highroad_winter_wp.pdf
30. EMC and Cisco NAS Solution - http://www.emc.com/partnersalliances/pdf/emc_cisco_network.pdf
31. EMC: Oracle 8 with Celerra Filer over NFS - http://www.emc.com/products/product_pdfs/wp/celerra/oracle8_cfs_nfs.pdf
32. Oracle Storage Compatibility Program (OSCP) White Papers - http://technet.oracle.com/deploy/availability/htdocs/oscp_papers.html#Oracle_OSCP
33. Administrating Oracle 9i on UNIX - http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/unix.920/a97297/ch1_admin.htm#i46647
34. NFS soft and hard mount - http://tclug.linux.org.tw/docs/ClientServer/tsld014.htm
35. NFS Mount – Soft and Hard mount - http://ou800doc.caldera.com/NET_nfs/nfsT.mount_cmd.html
36. NFS FS Advanced Mount Options - http://osr5doc.ca.caldera.com:457/NetAdminG/nfsD.nfsmount.html
37. Oracle 9i Fail Safe - http://otn.oracle.com/tech/windows/failsafe/content.html
38. Oracle Fail Safe on Compaq Proliant, 1998 - ftp://ftp.compaq.com/pub/supportinformation/papers/ecg050998.pdf
39. Commercial HA Software for Linux - http://linux-ha.org/commercial.html
40. Network Failover Strategies - http://linux-ha.org/failover/
41. Linux HA Heartbeat - http://linux-ha.org/comm/heartbeat-overview.html
42. Service Guard Commands - http://www.introcomp.co.uk/hpux/service_guard_commands.html
43. HP Business Continuity Solutions - http://h40056.www4.hp.com/storage/3_business_continuity.pdf
44. HP-UX 11i Single System Availability - http://www.hp-bg.com/pdf/Giga_HP_UX11i_1.pdf
45. HP Storage for Oracle and SAP - http://h40056.www4.hp.com/storage/4_solution_for_key_application.pdf
46. HP 1000Base-SX Product Tour - http://techsolutions.hp.com/dir_gigabit_external/training/a492456a.html
47. HP SureStore Virtula Array 7100 - http://www.rand.at/artikel/ts-news/hp_surestore/VA%207100.pdf
48. Compaq SAN - http://www.compaq.com/products/storageworks/msa1000/index.html

49. Compaq NAS - http://www.compaq.com/products/storageworks/b3000/index.html
50. Linux LVM Howto - http://www.sistina.com/lvm_howtos/lvm_howto.pdf
51. Sistina GFS Howto - http://www.sistina.com/gfs_howtos/gfs_howto.pdf
52. Edge Serving with Sistina's GFS - http://www.sistina.com/EdgeServingSistinasGFS.pdf
53. Oracle 9i Installation Guide - http://www.sistina.com/EdgeServingSistinasGFS.pdf
54. Building Highly Available Database Servers Using Oracle Real Application Clusters, Jack Cai, Oracle White Paper, Oracle, 2001 – RAC_Availability.pdf
55. Linux Software RAID – http://www.linuxdoc.org/HOWTO/Software-RAID-HOWTO.html
56. RAC vs. Microsoft – RACvsMicrosoft.pdf
57. Techincal Comparison of Oracle 9i Real Application Clusters vs. IBM DB2 UDB EEE v7.2, An Oracle White Paper February 2002 by Sashikanth Chandrasekaran & Bill Kehoe – http://otn.oracle.com/deploy/performance/pdf/ibm_db2.pdf
58. HP-UX disk Setup Using sdsadmin - http://www.dubois.ws/people/paul/sdsadmin.html
59. Oracle 9i RAC on HP-UX - http://otn.oracle.com/products/oracle9i/htdocs/9iRAChp.html#54
60. Oracle 9i RAC Cache Fusion Delivers Scalability, An Oracle White Paper by Sohan DeMel, February 2002 - http://otn.oracle.com/products/oracle9i/pdf/cache_fusion_rel2.pdf
61. Linux DPT Hardware RAID How-to BY Ram Samudrala, February 2002 - http://www.ram.org/computing/linux/dpt_raid.html
62. HP Disk System 2300 - http://www.hp.com/products1/storage/products/disk_arrays/disksystems/ds2300/infolibrary/ds2300.pdf
63. LVM for Linux by Heinz Mauelshagen - http://www.nondot.org/sabre/os/files/Partitions/LVM.PDF
64. LVM Question LV MAX 255.99GB - http://lists.suse.com/archive/suse-axp/2000-Mar/0130.html
65. Linux XFS Filesystem by SGI - http://oss.sgi.com/projects/xfs/
66. JFS for Linux by IBM - http://oss.software.ibm.com/jfs/
67. Design and implementation of Second Extended Filesystem in Linux by Remy Card et al - http://e2fsprogs.sourceforge.net/ext2intro.html
68. Test of Six Linux File Systems - http://aurora.zemris.fer.hr/filesystems/
69. Filesystems Howto by Martin Hinner - http://www.tldp.org/HOWTO/Filesystems-HOWTO.html
70. Using the ext3 filesystem in 2.4 kernels - http://www.zip.com.au/~akpm/linux/ext3/ext3-usage.html
71. Linux Ext3 Howto by Rajesh Fowkar - http://www.symonds.net/~rajesh/howto/ext3/toc.html
72. NAS Serial Benchmark Performance - http://www.nersc.gov/research/FTG/pcp/performance.html

73. M-VIA – A High Performance Modular VIA for Linux - http://www.nersc.gov/research/FTG/via/
74. M-VIA FAQ - http://www.nersc.gov/research/FTG/via/faq.html#q1
75. An implementation and Analysis of the Virtual Interface Architecture by Philip Buonadonna, Andrew Geweke, David Culler, 1998 - http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Buonadonna893/INDEX.HTM
76. Oracle 9i RAC Installation with NetApp Filer on Fujitsu-Siemens Primepower (Solaris 8) http://www.netapp.com/tech_library/3165.html
77. Oracle 9i RAC Installation with NetApp Filer in Linux Environment - http://www.netapp.com/tech_library/3164.html
78. Oracle 9i Database Online Documentation - http://docs.oracle.com/cd_a97630/index.htm
79. IEEE 802.3ae – 10GB Ethernet - http://www.ethermanage.com/ethernet/10gig.html
80. [6] Oracle 9i DBA Guide Release 2 (9.2) - http://docs.oracle.com/cd_a97630/server.920/a96521/create.htm#998107
81. Linux Performance Tuning Tools - http://linuxperf.nl.linux.org/links.html