hp-ux networking

august 2003

invent

technical white paper

forcibly unmounting nfs filesystems

By: Dave Olker, SNSL Advanced Technology Center
Contributor: Randy Saum, Global Solutions Engineering

table of contents

introduction

Network File System (NFS) is an industry standard file sharing protocol that allows the filesystems residing on one system (the server) to be seamlessly accessed from other systems (clients) across a network. When both the client and server systems are configured properly and performing well, NFS allows client systems to seamlessly access remote filesystem resources as if they resided on locally mounted disks.

But what about when things are not working well, such as when the NFS server system stops responding to requests, either because of a network partition being down, a system panic, or a catastrophic hardware failure? In these situations, NFS filesystems can appear "hung" on the client systems resulting in application hangs and loss of productivity. At times like these it may be beneficial to unmount these NFS filesystems until the server system recovers.

Of course, unmounting NFS filesystems may be less of a concern if your NFS servers reside in a highly available cluster (i.e. ServiceGuard, TruCluster, etc.) and can recover from hardware and software faults quickly. However, for those customers who do not employ this level of server protection, having a mechanism available to forcibly unmount "hung" NFS filesystems is highly desirable.

The purpose of this white paper is to describe the various tools and methodologies available to HP-UX systems administrators to combat the situation where NFS filesystems have become "hung" and need to be unmounted. In addition to covering the procedures available in currently supported versions of HP-UX, this paper describes the forcible unmount approach taken by other NFS vendors, as well as the plans to integrate official support for forcible filesystem unmounting in a future release of HP-UX. Finally, this paper discusses several preventative steps you can take when configuring your NFS environment to decrease the likelihood of encountering the problem of NFS filesystem that cannot be unmounted.

problem statement

Most attempts at unmounting hung NFS filesystems from an unavailable server fail with an error such as "Device busy." This typically occurs when client-side processes are referencing NFS files or directories on the server at the time it went down or when client-side processes continuously attempt to access NFS resources on the down server. These processes must relinquish any references to the directory and its files before the NFS unmount command will complete successfully.

Other system resources that count towards a filesystem being considered busy are things like buffer cache memory pages, memory-mapped files, and page cache memory pages. If these resources are associated with the NFS filesystem in question they will render the filesystem busy. Therefore, any processes that have used buffer cache, page cache, or memory-mapped resources associated with the NFS filesystem in question must relinquish control of these resources before the filesystem can be unmounted.

Unfortunately, in most situations the processes holding NFS resources cannot be simply killed because they are sleeping at an uninterruptible level in the kernel where they cannot receive SIGTERM or SIGKILL signals.

Similarly, there are times when an NFS filesystem cannot be unmounted even when the server is available because the client's kernel believes the filesystem is "busy" – even when there are no processes accessing the filesystem. Regardless of the cause, the end result is the client system is unable to successfully unmount an NFS filesystem.

The most common course of action in these situations is to reboot the client system, thus killing all running processes and releasing any remaining NFS and Virtual Memory resources. Of course, this is a drastic solution and one that customers prefer to avoid at all costs.

Given the disruptive nature and inherent downtime associated with rebooting client systems, it should come as no surprise that one of the most frequently requested enhancements submitted by NFS customers is the ability to forcibly unmount "hung" or "busy" NFS filesystems without incurring any client downtime.

While HP-UX does not currently provide a direct means of forcibly unmounting filesystems,[1] there are methods and procedures available that in some cases will allow you to successfully unmount "hung" filesystems without requiring a client system reboot. Likewise, there are precautions you can take to protect your HP-UX client systems from the situation where NFS filesystems are considered "busy" when no processes are accessing them.

[1] As of HP-UX 11.23 (a.k.a. 11i v2).

other vendors' solution

Several NFS vendors, including Sun Microsystems, have added support for a new forcible unmount feature to their operating systems. Sun introduced a new "**-f**" option to the umount(1M) command in Solaris 8, which instructs the client to forcibly unmount the filesystem regardless of whether any processes are accessing the filesystem or not.

This forcible unmount feature has the following characteristics and implications:

- The filesystem being forcibly unmounted simply disappears from the namespace
- Any existing processes using the filesystem are returned an I/O error (EIO)
- The system is prevented from consuming any new resources for the unmounted filesystem and any resources in use by the filesystem prior to the unmount request are cleaned up as much as possible
- Any locks held by the NFS client for files residing in the forcibly unmounted filesystem are released
- When a file I/O operation is attempted on a memory-mapped file in the unmounted filesystem the application will receive either a segmentation violation (SIGSEGV) or a bus error (SIGBUS)
- Any data being written to the unmounted filesystem that has not been committed would be lost
- The processes using this unmounted filesystem may not receive any indication that the filesystem has been forcibly unmounted

As the above list implies, the use of this option is not without potentially adverse effects to client-side applications. The potential exists for data loss, and any application using the filesystem being unmounted faces the possibility of aborting due to a SIGBUS or SIGSEGV error being returned from a pending I/O operation.

Of course, the same potential for data corruption exists if your only course of action is to reboot your NFS client systems, and most customers would prefer to avoid rebooting their systems whenever possible.

hp's future direction

Hewlett-Packard recognizes the potential benefits offered by this forcible unmount feature and we are considering adding this feature to a future OS release. As of the time of this writing, the target release for this feature is 11.31 (11i v3), but these plans could change at any time.

available solutions

While HP will not offer an officially supported forcible unmount solution for some time, there are several steps that you can take today to work around this problem. All of the solutions documented in this section should work on currently supported HP-UX 11.0 or 11i systems.

wait and try umount(1M) again

As stated earlier, the primary reason why NFS filesystems are considered "busy" and cannot be successfully unmounted is because client-side processes hold NFS resources in the specified filesystem and must relinquish these resources before the filesystem can be unmounted.

If an initial unmount attempt fails for an NFS filesystem, it is always a good idea to wait for a brief period of time (typically a few minutes) and try unmounting the filesystem again. It is always possible that a subsequent unmount command will succeed – if the processes waiting on the down NFS server either timeout or give up waiting for the server to recover and release the NFS resources they were holding. Also, during this unmount interval there is the possibility that buffer cache pages, memory mapped file pages, or page cache memory pages associated with the NFS filesystem in question could be freed and invalidated, thus allowing the filesystem to be unmounted successfully.

Of course, if the applications holding the NFS resources are not designed to timeout and give up you could find yourself waiting forever for something that won't happen.

kill any processes accessing the nfs filesystem

While certain processes holding NFS resources sleep at an uninterruptible level in the kernel and cannot receive signals, many times the processes that are keeping an NFS filesystem busy can be successfully killed, thus allowing the filesystem to be unmounted. The easiest way to determine which processes are keeping an NFS filesystem busy, and whether these processes can be successfully killed, is to use the fuser(1M) command.

The fuser(1M) command, when issued against a file, lists the process IDs of any processes that have the specified file open. When issued against a filesystem, fuser(1M) lists the process IDs of all processes that currently have open files residing in the specified filesystem. fuser(1M) can also be used to programmatically send kill signals to these identified processes.

Figure 1 on page 6 shows an example of using the fuser(1M) command to successfully identify and kill the processes holding an NFS filesystem busy. The various steps illustrated in Figure 1 include:

1. Attempt to unmount the NFS filesystem via the umount(1M) command

2. Use the fuser(1M) command to identify the process IDs of those processes that have files open in the specified filesystem

3. Use the ps(1) command to view the offending processes. The "-p" option instructs ps(1) to only return information for the specified process IDs.

4. Use the fuser(1M) command with the "-k" option to kill the processes holding files open in the target filesystem. (See important fuser(1M) syntax note below.)

5. Issue the umount(1M) command again to successfully unmount the NFS filesystem now that all processes holding files open in the filesystem have been killed.

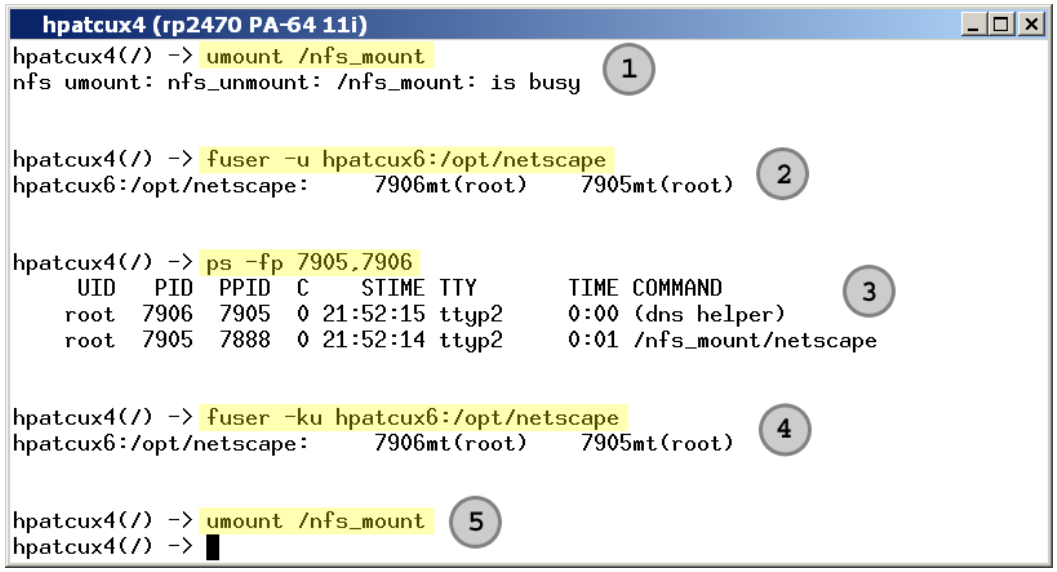

Figure 1 - fuser(1M) successfully killing processes holding an NFS filesystem busy

In the above example, fuser(1M) was able to correctly identify and kill the processes that were holding the NFS filesystem busy, thus allowing the filesystem to be successfully unmounted while the NFS server was down.

important note on fuser(1M) syntax with NFS filesystems

When using fuser(1M) to query NFS filesystems, always specify the target filesystem using the format *"server:/filesystem"* as opposed to specifying client-side path where the filesystem is mounted. For example, in the case of the following NFS filesystem:

```
Filesystem                                    Mounted on
hpatcux6:/opt/netscape                        /nfs_mount
```

The target filesystem specified on the fuser(1M) command line should be **hpatcux6:/opt/netscape** and not **/nfs_mount**.

fuser(1M) is able to recognize the *server:/path* syntax as being an NFS filesystem and it makes no attempt to *stat()* the remote filesystem, which in the case of a down NFS server would result in an fuser(1M) hang.

setup a temporary "surrogate" nfs server

In the example shown in Figure 1 the fuser(1M) command was able to kill the processes holding the NFS filesystem busy, and thereby allow the filesystem to be unmounted. However, in many situations the processes accessing a down NFS filesystem are sleeping at an uninterruptible level in the kernel and are unable to receive signals, such as SIGKILL or SIGTERM. In these situations, fuser(1M) will not be able to kill these processes and the filesystem will remain busy.

Obviously the best solution in this situation is to get the real NFS server back in operation so that these blocked processes can complete their transactions. Unfortunately, it isn't always possible to bring the NFS server back online quickly, or at least not in an acceptable timeframe for NFS client users.

At times such as this, one possible solution is to setup a temporary "surrogate" NFS server. In other words, locate a working NFS server system in your environment and temporarily configure the IP address (or addresses) of the down server to this working NFS server. By doing this, the NFS client systems will be able to get a response from a live NFS server system. If an alternate NFS server cannot be located, another possibility would be to take one of the blocked NFS client systems and enable NFS server-side services on this system and then add the down NFS server's IP address to this system.

Figure 2 below and Figure 3 on page 8 illustrate the various steps involved in setting up a temporary "surrogate" NFS server. Each step in the example is numbered and described in detail below.

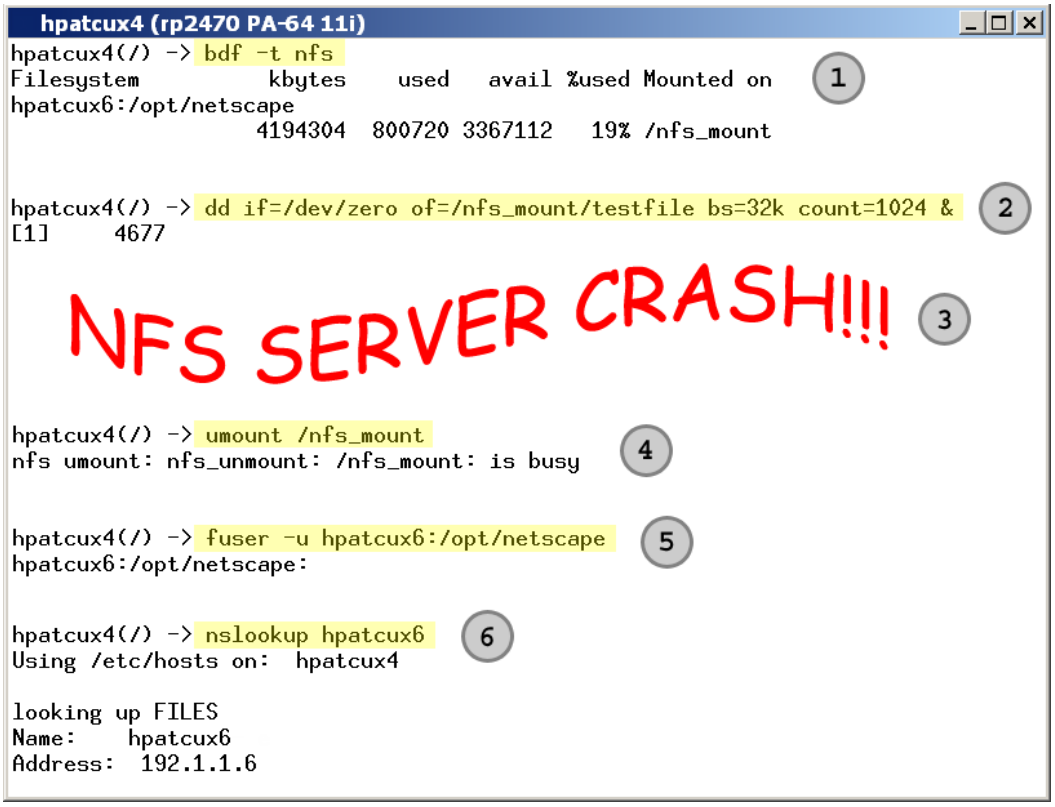

Figure 2 - Setup a "surrogate" NFS server (part 1)

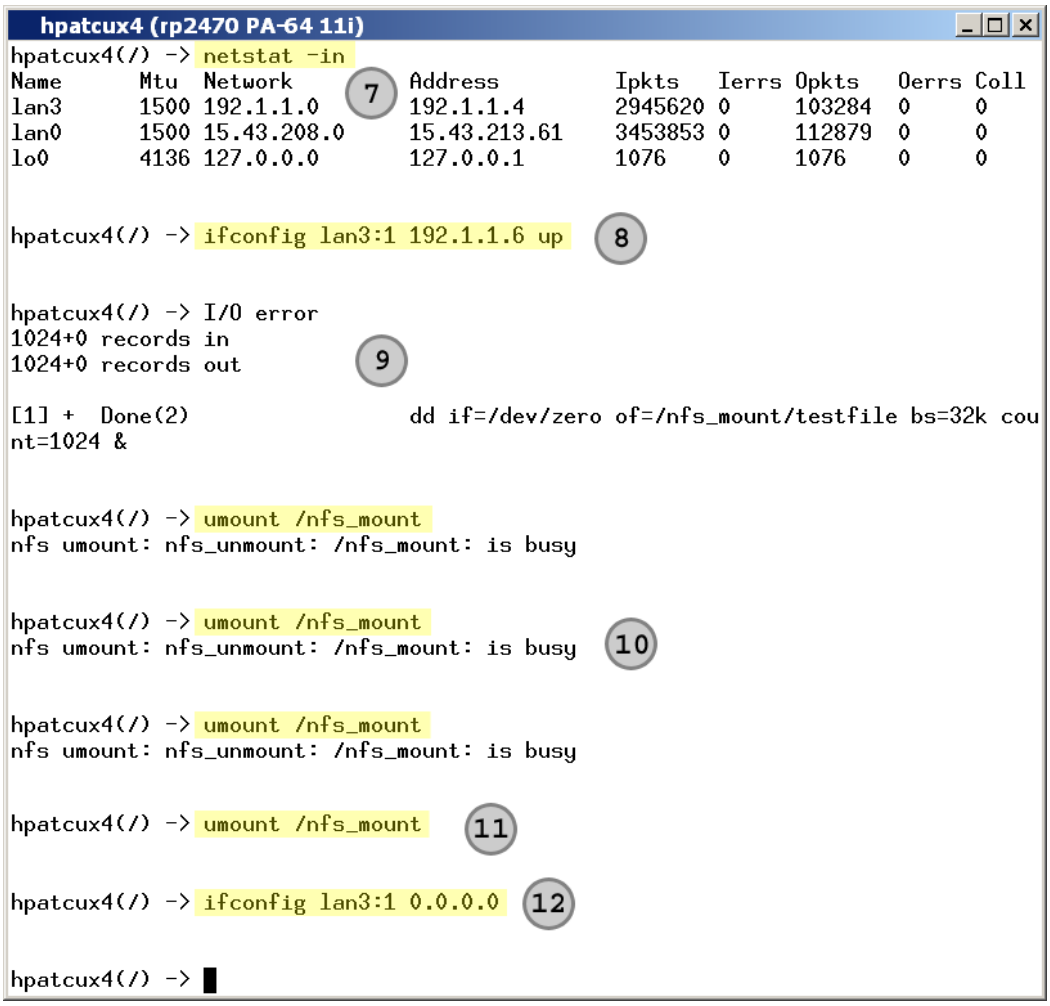

Figure 3 - Setup a "surrogate" NFS server (part 2)

The following steps were used in the above example to simulate an NFS filesystem hang scenario and then to create a "surrogate" NFS server:

1. The bdf(1M) command shows the mounted NFS filesystems.

2. The dd(1) command is used to generate file I/O in the target NFS filesystem. In this example, the dd(1) command is writing to a file in the NFS filesystem.

3. At this point, the NFS server crashes and the dd(1) command hangs.

4. Because the dd(1) command did not complete and has outstanding I/O requests requiring a response from the server, the NFS filesystem is considered busy and cannot be unmounted.

5. The fuser(1M) command is used to identify any processes with files open in the target filesystem. In this example, fuser(1M) reports that no processes have files open in the specified NFS filesystem.

At first glance this fuser(1M) output doesn't seem accurate since we know that the dd(1) command launched in step #2 is referencing a file in the target filesystem. Why then does fuser(1M) not report the dd(1) process as having open files? The reason for this apparent discrepancy is that the outstanding NFS write requests for this file are queued in the client's buffer cache memory waiting to be written to the server, so the server's file is not technically "open" at this point. However, these buffer cache pages count against the client's overall usage of the NFS filesystem, so it is these buffer cache pages that are keeping the filesystem busy.

Even if fuser(1M) had been able to identify the dd(1) process as the one holding the target NFS filesystem busy, fuser(1M) would not have been able to successfully kill this process because dd(1) was in the middle of performing file I/O operations at the time the server crashed and it would therefore be sleeping at an uninterruptible level in the kernel – unable to receive signals like SIGKILL and SIGTERM. Even manually sending a "kill -9" to this process would have no effect, given the state the process was in.

At this point we have a client NFS filesystem that is hung and cannot be unmounted until it gets a response from the NFS server. Since the original server is unavailable and presumably cannot be restored in a timely manner, the alternative solution is to setup a "surrogate" NFS server.

6. The first step in creating a "surrogate" server is to determine the IP address of the down NFS server. In this example, the nslookup(1) command is used to retrieve this information.

7. Now a suitable replacement system must be found. The system needs to be running NFS server daemons (i.e. nfsds). In this example, we will use the NFS client system, which happens to be running nfsds, as the "surrogate" server.

   Before configuring the server's IP address on this system, we first need to determine which IP interface to plumb the server's address to. The netstat(1) command is used to display the configured IP interfaces on the surrogate system. Examining this output, it appears this system has three IP interfaces: lan3, lan0, and lo0. The lan0 interface is connected to the 15.43.208.0 subnet, lan3 is connected to the 192.1.1.0 network and lo0 is the loopback interface. Since the down server's IP address is a member of the 192.1.1.X network, lan3 is the appropriate interface on this system to plumb this address to.

8. The ifconfig(1M) command is used to add the server's IP address to the client's lan3 interface.

9. Almost immediately the dd(1) command reports an "I/O error" and exits. This is expected behavior since the temporary NFS server (i.e. the client in our example) is not exporting the same filesystems as the original server, so the NFS requests for the original target file will be considered "stale" and will be responded to with an ESTALE error. This ESTALE error indicates to the client dd(1) process that the file it was referencing no longer exists on the responding server.

Depending upon the design of the application, most processes, upon receiving an ESTALE error, will give up attempting to contact the NFS server and will either exit on their own or will transition to a state where they can be successfully killed. In this example, the dd(1) application returned an error and exited.

10. Even though the dd(1) command has exited, the filesystem cannot be immediately unmounted. Remember that the client's buffer cache was holding memory pages associated with the target NFS filesystem. These pages must be invalidated from the client's cache before the filesystem is considered "not busy."

    This is a good example of the principle described on page 5 – if an initial umount(1M) attempt fails, wait for a period of time and try again. In this example, the final working umount(1M) command occurred approximately 2 minutes after the dd(1) command exited. Of course, the actual time it takes for your applications to relinquish their NFS resources will vary.

11. Eventually all buffer cache memory pages associated with the NFS filesystem are invalidated and the filesystem can be successfully unmounted.

12. Once all NFS clients have successfully unmounted any filesystems associated with the "down" server, the ifconfig(1M) command should be used to un-plumb the original server's IP address. This step is important to remember in order to avoid an IP address conflict once the original NFS server returns to normal operation.

This procedure is by no means foolproof, nor is it guaranteed to work in every situation. However, by configuring an available system to masquerade as the down NFS server, there is a good chance that most NFS client processes will eventually give up waiting for the original server and relinquish their resources, allowing hung NFS filesystems to be successfully unmounted until the original server can be made available again.

preventative steps

As discussed earlier in this paper, there are many reasons why an NFS filesystem may at some point transition to a state where it cannot be unmounted; such as when processes are holding resources open on the filesystem and cannot be killed. There are also times when it appears no processes are holding any NFS filesystem resources and yet the filesystem is still considered "busy." We saw in the example beginning on page 7 how a process can have active buffer cache memory resources pending against an NFS filesystem and how this process may not show up in an fuser(1M) process list even though it continues to hold these buffer cache pages. There have even been customer reported cases where no processes are keeping NFS files open or holding buffer cache resources against an NFS filesystem, yet the filesystem is still considered "busy."

Whatever the cause, the result is the same – an NFS filesystem that cannot be unmounted when desired. While some of these "busy" NFS filesystem scenarios are unavoidable, such as server hardware or software failures, there are precautions you can take when configuring your NFS clients and servers to avoid many of the other common situations that cause NFS filesystems to be incorrectly considered "busy." Even in the case of server hardware or software failures, there are steps you can take to protect your NFS clients from prolonged server outages.

install the latest fuser(1M), NFS, and virtual memory patches

Periodically, NFS filesystems may remain in a "busy" state even though no processes that are accessing this filesystem can be identified by fuser(1M). This situation could be caused by any number of reasons – processes holding valid buffer cache resources but do not have files open in the NFS filesystem,[2] a defect in the fuser(1M) code that hinders its ability to report processes correctly, or defects in the NFS or Virtual Memory subsystems that cause NFS filesystem data structures to become inaccurate, causing the kernel to incorrectly consider a filesystem "busy" when it is really idle.

HP has released several patches to address these problems on HP-UX 11.0 and 11i systems. The patches listed below in Table 1 were current as of the time of this writing; however these patches may have since been superseded. In addition, several of the patches listed in Table 1 are dependent on other patches for proper operation. Before installing any of these patches on your HP-UX NFS clients and servers, check with HP Support to obtain a current list of patches for your specific operating system, or use the patch tools available at HP's IT Resource Center Web site: http://itrc.hp.com.

Table 1 - Recommended Patches to Avoid Known NFS umount(1M) Problems

Patch Name	Supported OS	Patch Description
PHCO_21901	11.0	fuser(1M) cumulative
PHNE_28567	11.0	ONC/NFS General Release/Performance
PHNE_28568	11i	ONC/NFS General Release/Performance
PHKL_27266	11i	iCOD, RTSCHED, (u)mount, final close, NFS umount

[2] Refer to the example shown on pages 7 through 10.

use the "soft" NFS mount option

By default, NFS filesystems are mounted with the "hard" option, which instructs the client's kernel to indefinitely retransmit any NFS request that is not responded to by the NFS server. In other words, if a process on an NFS client attempts to access a file on an NFS server that is down or otherwise unresponsive, the client will continue to resend NFS requests to the server until that system (or a different system masquerading as the server)[3] responds. This behavior is intentional and designed to guarantee data integrity.

While there are definite benefits to this "hard" mount behavior, there are also drawbacks – especially in environments where NFS servers are frequently unavailable. In these environments it may be preferable to allow NFS clients to eventually "give up" when trying to communicate with a down server and return control to the requesting applications. This can be done by mounting the NFS filesystems using the "soft" option.

In the previous example on pages 7 through 10, the NFS filesystem was mounted with the default "hard" option, forcing the client to continually resend NFS requests to the server until it received a response. If a "surrogate" server had not been configured, or the original NFS server not been restored to service, any client applications accessing this filesystem would hang indefinitely.

In Figure 4 below the same exercise is repeated, except in this case the NFS filesystem is mounted with the "soft" option. When the NFS server crashes (in step 3) the client's kernel will attempt to communicate with the server by retransmitting the NFS requests that have not been responded to. However, the "soft" option instructs the client to eventually give up and return an I/O error to the dd(1) process. Once all of the processes accessing this hung NFS filesystem exit the filesystem can be successfully unmounted.

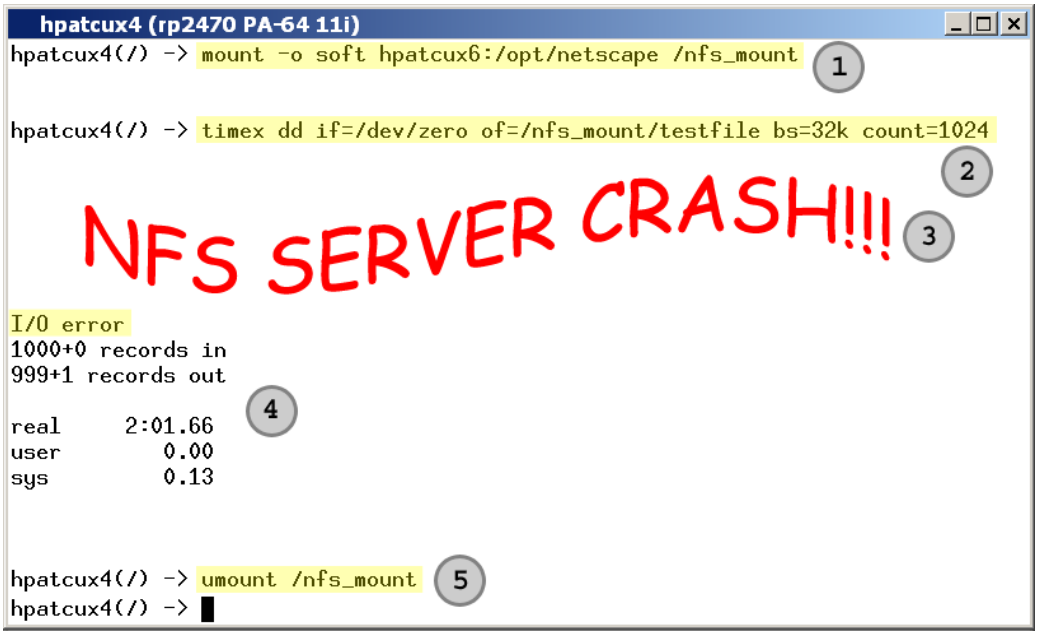

Figure 4 – Writing to a down NFS server via a "soft" mounted filesystem

[3] Refer to the example shown on pages 7 through 10.

important note on the "soft" NFS mount option

Use of the "soft" option on NFS filesystems mounted for read/write access can be dangerous if your applications are not designed to gracefully handle receiving a timeout error for operations such as *read()* or *write()*. With certain applications, allowing an NFS *write()* call to return an I/O error can lead to data corruption if the client application fails to check the return status of its *write()* calls and mistakenly assumes that its data has been successfully written to the server when in fact the *write()* call timed out. For this reason, the "hard" mount option (default) is recommended whenever any *write()* operation will be performed on the mounted NFS filesystem.

If your NFS environment is one where server systems are frequently unavailable or non-responsive and you know that all of your client-side applications are designed to properly handle receiving an I/O error in response to a *read()* or a *write()* call, the "soft" mount option may be a viable means of alleviating some of the frustration and downtime associated with hung filesystems.

use ServiceGuard to setup a highly available NFS server environment

One of the reoccurring themes throughout this paper is the idea that applications accessing an NFS filesystem that is mounted with the "hard" option (default) will hang indefinitely if the server becomes unavailable during their I/O attempt. In the example on pages 7 through 10, a procedure was outlined showing how a secondary NFS server can "masquerade" as the real NFS server. This temporary server receives the retransmitted NFS requests from the clients and returns an ESTALE error because it is not managing the same filesystems exported by the original server.

While this ESTALE error does allow the hanging client applications to unblock and continue (or in some cases exit), a more desirable outcome would be if the secondary NFS server could somehow export the same filesystems as the original server and thereby take over responsibility for these NFS filesystems while the original server is unavailable. If this were to occur, the client applications would operate normally despite the fact that their NFS filesystems had migrated between servers, and while they may experience a temporary interruption while the filesystems actually migrated, they would quickly be able to seamlessly continue their operations with the new server.

This scenario is available today on HP-UX 11.0 and 11i using HP's MC/ServiceGuard product and Highly Available NFS Server component.

MC/ServiceGuard allows you to create high availability clusters of HP 9000 servers. A high availability computer system allows application services to continue in spite of a hardware or software failure. Highly available systems protect users from software failures as well as from failure of a system processing unit (SPU), disk, or local area network (LAN) component. In the event that one component fails, the redundant component takes over. Application services (individual HP-UX processes) are grouped together in *packages*; in the event of a single service, node, network, or other resource failure, MC/ServiceGuard can automatically transfer control of the package to another node within the cluster, allowing services to remain available with minimal interruption.

The Highly Available NFS component of MC/ServiceGuard is a toolkit that enables you to create NFS packages that run on highly available servers. With MC/ServiceGuard NFS, an NFS server package containing exported filesystems can move from one node (the *primary* node) to a different node (the *adoptive* node) in the cluster in the event of failure. After MC/ServiceGuard starts the NFS package on the adoptive node, the NFS filesystems are re-exported to the clients. Any client-side applications accessing these filesystems will hang temporarily while the NFS server package is started on the adoptive node. Once the package is running on the adoptive node, the client-side applications can continue to seamlessly access their files without having to be restarted.

Looking back at the example on pages 7 through 10, remember that one of the critical steps in setting up a temporary "surrogate" server involved configuring the IP address of the original NFS server on the temporary node. MC/ServiceGuard NFS performs a similar function when migrating NFS packages between nodes in the cluster.

Each highly available NFS package is assigned a dedicated IP address, and it is this IP address that the NFS clients use when mounting the exported filesystems associated with the package. When a hardware or software failure occurs on the primary node, and this failure causes MC/ServiceGuard to decide that a package migration is needed, the exported filesystems and IP address associated with this NFS package move to an adoptive node in the cluster. Since both the exported filesystems and the server's IP address migrate between systems, NFS clients do not require any special configuration to work in this environment because they simply retransmit their NFS requests to the same IP address both before and after the package migration. The client systems have no idea that a different server node in the MC/ServiceGuard cluster is now running the NFS package and is responding to their requests.

By employing a highly available cluster of NFS servers that can share responsibility for a pool of exported filesystems, the likelihood of encountering a situation where NFS clients are blocked due to an unavailable NFS server is drastically reduced.

summary

There are many times when an NFS filesystem will be in a state where it cannot be unmounted. In some situations these filesystems are considered "busy" for legitimate reasons – such as when processes are actively accessing the filesystem or when one or more processes are holding buffer cache memory resources that reference the filesystem. At other times a filesystem may be erroneously deemed "busy," where the filesystem cannot be unmounted even though no processes are accessing or referencing it.

Whatever the cause, when a filesystem cannot be successfully unmounted and the NFS server system stops responding to requests, either because it has suffered a hardware or software failure or because it is simply overwhelmed with requests, the result can be client application hangs or similar interruptions of service, leading to user frustration and loss of productivity. At times such as these, it may be desirable to find a way to remove these filesystems from the client until the NFS server's responsiveness can be restored. While HP does not currently provide a mechanism to directly force a client to unmount a "busy" or "hung" NFS filesystem in HP-UX 11.0 or 11i v1/v2, HP is considering providing this functionality in an upcoming release of HP-UX.

Until such time as a forcible unmount feature is supported by HP-UX, there are many steps you can take in your current environment to alleviate the negative impact of these "hung" filesystems. In most cases the fuser(1M) command can be used to identify and kill the processes that are accessing or referencing the filesystem, thus allowing the filesystem to be unmounted. When this fails, a temporary "surrogate" NFS server can be configured to return an error to the clients, which usually causes the client applications to release their hold on the "hung" filesystem. Sometimes simply waiting for a period of time and trying the umount(1M) command again may succeed.

In addition, there are preventative measures you can take to avoid experiencing this "hung" filesystem problem. HP has released software patches that address defects in our NFS client kernel code and virtual memory subsystem that erroneously caused filesystems to be deemed "busy" when there were no legitimate users of the filesystem. There are also patches for the fuser(1M) command that address problems where fuser(1M) was not correctly identifying all processes accessing a mounted NFS filesystem. The "soft" NFS mount option can be used (with caution) to allow client applications to eventually give up trying to contact an unresponsive NFS server. Finally, HP's MC/ServiceGuard and Highly Available NFS products can be used to create a cluster of highly available NFS servers, which can alleviate the problem of unresponsive NFS servers entirely.

for more information

To learn more about HP's MC/ServiceGuard and Highly Available NFS products, contact your local HP sales representative or visit our Web site at: http://www.hp.com/go/ha. For information about HP's NFS product family or the various commands used throughout this paper, visit HP's Documentation Repository Web site at: http://docs.hp.com or HP's IT Resource Center Web site at: http://itrc.hp.com.