ND Elimination Strategy

Bob Lyon

1. Introduction

The next major task for the NFS group is to eliminate the dependence of diskless Suns on the ND product. Elimination of ND is desirable due to the fact that ND is wasteful of disk space and is impossible to administrate in a dynamic environment. The biggest problem with ND elimination is the loss of performance; this problem is not addressed in this document, though the group already has ideas as to how to keep the performance as fast as ND.

There are three major tasks involved in running diskless machines entirely off of the NFS; they are:
(a) the client's ability to swap to an NFS based file,
(b) the client's ability to get a root file system from an NFS server, and
(c) the client's ability to boot from a tftp (rather than ND) based server.
Since (c) is the most difficult and since it must be accomplished by release 3.0, the document concentrates primarily on this task.

2. Diskless Architecture

Diskless clients continue to have default servers which contain the clients' root file systems and boot files. As with ND, these defaults can be changed manually at boot time.

A server of root file systems will export a file system with the following two new features: (a) uid 0 (root) is *not* mapped to some other uid, and (b) the server never gives out the fhandle associated with the file system's root inumber ("inode 2"). Furthermore, the access control list feature of the /etc/exports file (employed by the rpc.mountd daemon) will be utilized to insure that only the specific client machine may import its corresponding subtree of the "roots" file system.

For example, assume the server machine titan has two disks and four clients — gaia, oriskany, ganymede, and pluto. Some of the entries in /etc/fstab may look like:

```
/dev/xy0g /roots1 4.2 rw 1 3
/dev/xy0d /roots2 4.2 rw 1 4
```

The associated /etc/exports file may look like:

```
/roots1             nobody
/roots1/gaia        gaia
/roots1/ganymede    ganymede
/roots2             nobody
/roots2/oriskany    oriskany
/roots2/pluto       pluto
/usr
/usr/titan
/usr/doctools
```

Note that the last three lines represent features which are utilized in release 2.0 and essentially allow any machine to mount the file systems. The first six lines are allowed (but seldom used) in release 2.0. They effectively limit who may mount which subtree of the roots file systems.

In addition, /etc/exports should have new lines which enable the new features of the NFS servers; the features are associated with file systems. These features include netroot mapping, file system root fhandle restriction, and permission masks. For example an entry may look something like (the syntax obviously needs work):

```
/roots1          root->0, no_root_fhandle, 7777
```

The combination of (a) allowing root access, (b) never handing out the fhandle for inode number 2, and (c) the placement of clients' roots within the server's file system's root directory guarantees secure, exclusive (though restricted) access of clients to the server.

The exclusive access to subtrees is similar to ND. This differs from ND in that the file system itself (particularly, the file system's free space) is shared among all the clients.

2.1. /pub disappears but its spirit remains

Instead of mounting a second ND-based file system on /pub, the client machines now mount a "bin" NFS-based file system on /bin. Also, since most the components of /etc are executables, they could be moved to /bin/etc. Compatibility can be maintained either by modifying shell scripts that depend on executables being in /etc or with symbolic links or some combination of the two. For example, if pluto were to boot in a default manner, it would mount the following when coming up single user:

"titan:/roots2/pluto" on mount point "/"
"titan:/68020bin" on mount point "/bin"

As in the spirit of ND based /pub file systems, titan can now export its /68020bin file system as read only and it need not allow root access to it. Titan most likely has another exported file system called /68010bin. The client's mount points "/" and "/bin" are built into the diskless kernel and are not variable.

2.2. ps, vmunix, and tftp booting

The elimination of ND affords us the oppertunity to make booting client specific vmunixes easy. That is, client's private vmunixes should be automatically rebooted when the client system crashes. However, as with the existing ND based architecture, the sharing of vmunix should continue to be easy.

The tricky part is to make the server and the client agree upon which vmunix file is being booted. Booting details are discussed later. Suffice it to say here that as distributed in release 4.0, pluto will always tftp boot the file "titan:/roots2/pluto/vmunix".

Assuming pluto is powered down makes discussing its file system on titan easier. Pluto (or any other client) has an entry in / of:

```
vmunix -> bin/vmunix
```

Note that there is no / in front of bin, but from pluto's point of view the two are equivalent. This is not the case for titan since pluto's / is really titan:/roots2/pluto.

As discussed in section 2.1, pluto's "/bin" is really a mount point directory, *but* it also has one entry to assist tftp booting; this entry is:

```
vmunix -> /68020bin/vmunix
```

In this case, the symbolic link's target is absolute on the tftp server (titan). The tftp boot file happens to correspond to pluto's /bin/vmunix once pluto mounts its /bin file system.

Note that installing a custom vmunix is as hard as it is today under ND. The user can't just cp the new onto the old since the old is a symbolic link to a file in a read-only file system; he must rm /vmunix, then mv or cp the new one in place. Unlike ND however, the new system will

automaticly reboot the correct vmunix after every kernel crash. This means that savecore, ps and perfmeters will always work. This feature is very valuable to dumb users (like the author).

This hack does not generalize well but it is worth including as the default setup.

2.3. swapping to an NFS file

This is trivial. Each client machine has a new file called "/dev/swap" in his private subtree. Thus pluto would swap to his "/dev/swap" file; titan would see the file as "titan:/roots2/pluto/dev/swap".

The swap files are staticly created (and allocated) at setup time. Unlike ND, the swap files can be grown at any time if and when more swap space is needed by a particular client. The easiest way is to

```
cat bigfile >> /dev/swap
```

In this manner less disk space will be wasted due to pre-allocation of unneeded swap space.

3. PROM Booting

This section discusses what the prom must do in order to boot a Sun machine without the help of ND. Clearly the diskless tasks must be integrated with diskful tasks, although this memo only is concerned with diskless booting. The section references services described in future sections.

The prom needs to tftp boot the booter program. Since tftp is based upon IP, the prom must first discover the machine's IP address via the reverse-ARP (RARP) protocol. (This protocol is broadcast based; an RARP server will map the broadcast's source address (48 bits) to its appropriate IP address (32 bits). The design of the RARP server is discussed in a later section.)

Next the prom must decide what file to tftp boot and from what server. Upon finding its IP address, the machine turns the address into a suitable string which represents a file name. For example, pluto would generate the string "192.9.1.17" and use this as the name of his boot file. (Can tftp take relative, rather than absolute names?) The server can be located by broadcasting for the first block of the file. The client may try to avoid broadcasts by first trying to get the file from the machine that originally answered the RARP request.

The file name and server location algorithm are desirable because they are easy to document and depend only upon existing ARPA standards -- tftp and RARP.

Finally, the client must run the ARP protocol both as a client and a server. This solves the problem of the server machine being rebooted when the client is booting from it. (The boot server tries to send a response, but cannot unless the client responds the boot server's ARP request.) The problem currently exists with ND.

3.1. Complications

The machine should be able to boot even when it is connect to more than one network interface. This does not change any of the specification.

There should be some sort of command option that allows the user to specify (override) the boot file name.

Various people feel that new machines should be able to boot (standard diagnostics) without convincing the system administrator to add new entries to /etc/hosts and /etc/rarp.

4. /boot Booting

The section describes what /boot must do in order to get a typical vmunix booted on a diskless Sun machine. Clearly the diskless tasks must be integrated with diskful tasks, although this memo only is concerned with diskless booting. The section references services described in future

sections.

5. vmunix Booting

This section discusses what vmunix has to do in order to get up to single user mode. The section references services described in future sections.

6. Network Services That Assist Booting

This section discusses the new network services that are necessary in order for tftp booting to be accomplished.