1 Filesystem Setup and Maintenance

Introduction

This module discusses the LFS filesystem, in particular how to create, check, and mount such filesystems. NFS exporting of LFS filesystems is then covered.

Objectives

By the end of this lesson, you will be able to:

- Create LFS filesystems
- Check LFS filesystems for consistency
- Mount LFS filesystems
- Share filesystems via NFS
- Set appropriate restrictions on shared filesystems
- Handle degraded filesystems

Filesystem Overview

LFS filesystems implement the High Throughput Filesystem (HTFS)

mkfs and fsck now both run entirely on the FSP

Filesystem consistency problems are handled by filesystem degradation, which renders a filesystem (or a single file) read-only

Filesystem Overview

Filesystem Overview

The M2000 utilizes LFS filesystems on the FSP for optimized filesystem performance. LFS is actually an implementation of the High Throughput Filesystem (HTFS), which provides performance superior to the standard BSD UFS filesystem.

Filesystems are created with mkfs, which runs on the FSP. Filesystem consistency checks, fsck's, are also run on the FSP. Filesystems are mounted to an FSP, and then shared for NFS access.

In order to handle file system consistency problems, LFS filesystems can enter a degraded state. This is similar to the filesystem isolation functionality of the NetServer product, but degradation may occur on individual files rather than entire filesystems, and renders the filesystem read-only rather than completely inaccessible.

HTFS Journaling

Filesystem journaling allows shorter fsck times after an abnormal system halt

Journaling groups filesystem operations for efficiency and writes these operations to an intent log before committing them to disk

This log can be replayed during crash recovery, and only data being acted upon need be checked (fast fsck)

HTFS Journaling

High Throughput Filesystem

LFS filesystems are actually implemented as a High Throughput Filesystem (HTFS). This new filesystem offers greater performance as well as advanced features including filesystem journaling and filesystem degradation. HTFS is also extensible, meaning that functionality can be added to it without requiring a complete overhaul.

Journaled Filesystem

The HTFS filesystem implementation creates transaction groups of filesystem operations which maximize disk access efficiency. As these are committed to disk an “intent log” is updated to identify the intent of the operations to follow. If the system is halted abnormally, this log can be "replayed" and transactions are either completed or aborted. As will be detailed later, fsck takes advantage of this functionality to greatly decrease the time required for filesystem checks.

Filesystem Degradation

In order to handle filesystem consistency errors which might lead to corrupted data, HTFS supports filesystem degradation, which is similar to the isolation feature of the Auspex NetServer, but is less obstructive.

When a filesystem inconsistency is detected, the extent of the inconsistency determines whether a file or the entire directory is degraded. If the data which appears inconsistent only affects a single file, then that file will become read-only and a message will be logged to the system logs and the console (assuming syslog.conf is set to do so; see syslogd discussion in the Performance module).

If the data inconsistency deals with directory-level metadata (anything beyond a single file), the entire filesystem will be rendered read-only. Again, a message will be sent to the system logs and the console.

Regardless of the severity of the failure, the filesystem will need to be manually fscked to resolve the problem. Note also that if there is a disk error which prevents filesystem data access, as in the case of a RAID7

HTFS Filesystem Degradation

Filesystem degradation replaces isolation

Degradation may occur on a single file (if a single file is affected) or the entire filesystem

As with isolations, disk failure affecting filesystem accesses will cause degradation

An error message regarding the filesystem degradation will be logged via syslogd

HTFS Filesystem Degradation

array which loses a member, the filesystem will also degrade; this is similar to the isolation functionality of the current NetServer product, which also isolates filesystems which experience disk problems. Filesystem degradation due to disk errors implies that there are no redundant disks, so the error must either be resolved on that disk (for instance, reassigning sectors) or the disk will have to be replaced and the filesystem restored from tape.

Filesystem Creation

"mkfs -F lfs" will trigger a process to create a filesystem on the FSP

mkfs will report the required size of the fsck scratch partition for the newly created filesystem

Slidetitle

Creating a Filesystem

LFS filesystems are created with mkfs, which initiates a process on the FSP to create a filesystem on the indicated logical volume. The mkfs command has the following syntax:

```
mkfs -F <fstype> [<options>] <device>
```

fstype should be lfs for all filesystems which will be shared over NFS. device refers to either a raw RAID device (such as /dev/raxrd/fsp0m0rd1) or a raw virtual partition (such as /dev/raxvp/fsp0vp1). No options need be specified at this point in time (refer to the generic mkfs(1) manpage for possible options). The system will respond as follows:

```
piaget# mkfs -F lfs /dev/raxvp/fsp0vp0
fsp0vp0:

Size: 17365MB (35565552 sectors)
Layout: 136 group(s) (128MB per group)
Blocks available: 35555455
Maximum files available: 127999584

You will need a scratch device of size 18 MB for full fsck of fsp0vp0
```

The scratch device mentioned will be covered below.

Filesystem Consistency Checking

The fsck process runs entirely on the FSP once initiated from the command line on the host

fsck requires a fsck scratch partition for metadata it is checking

FSP-based fscks only work with LFS filesystems

Host-mounted UFS filesystems are checked with the standard Solaris fsck

Filesystem Consistency Checking

Checking Filesystem Consistency

Filesystem checking, performed with the fsck command, is implemented entirely on the FSP in the M2000. As such, the FSP must have space for temporary storage of metadata on which it performs consistency calculations, and this is provided with a scratch partition. The FSP-based fsck is usually able to utilize the intent log to achieve short fsck times. Note that FSP-based fsck's only work with LFS filesystems. Filesystems mounted on the host will be checked with standard Solaris utilities.

Creating Scratch Partitions for fsck

Scratch partitions are RAID array partitions or virtual partitions, assigned per FSP. When a filesystem is created with mkfs, mkfs will report the required scratch partition size for fsck'ing the new filesystem.

Each FSP must have one, and only one, scratch device. If multiple partitions are specified, the first will be used and the rest ignored.

Scratch partitions are specified in /usr/AXbase/etc/fscktab, which is simply a listing of partitions which can be used by fsck:

```
/dev/raxmrd/fsp0m0rd0s1  # fsck scratch for FSP0
/dev/raxvp/fsp1vp2       # fsck scratch for FSP1
```

Note that these are the raw (character) devices rather than the cooked (block) devices (hence raxmrd and raxvp rather than axmrd or axvp).

Note: Scratch partitions must not be mounted or part of a virtual partition. They do not and will not contain a filesystem, so it is not necessary to run mkfs.

fscktab is read by ax_fsck_scratch, which runs at boot-time, but should also be run by hand whenever you change scratch partitions. It has the following syntax:

- ax_fsck_scratch -s
  Set the scratch devices listed in /usr/AXbase/etc/fscktab
- ax_fsck_scratch -r
  Release the scratch devices set on all FSPs

fsck Scratch Partitions

Used to contain metadata for consistency checking calculations

May be a virtual partition or RAID slice

Must not be mounted or be part of another virtual partition

Will not contain a filesystem, so running mkfs is unnecessary

Must be a raw device (raxmrd or raxvp)

Exactly one scratch device must be defined per FSP, and it must be large enough to handle the largest filesystem on that FSP

The ax_fsck_scratch command, in conjunction with the file /usr/AXbase/etc/fscktab, sets (-s), lists (-l), and removes (-r) fsck scratch partitions

fsck Scratch Partitions

▲ ax_fsck_scratch -l
List the scratch devices set on all FSPs

After running ax_fsck_scratch with the -s flag to read in fscktab, use the -l argument to verify that the devices are set as you intended:

```
# ax_fsck_scratch -l
FSP0 - fsp0m0rd0s1
FSP1 - fsp1vp2
```

Executing fsck

Filesystem consistency checking is performed by the fsck utility, which initiates a fsck process on the FSP for LFS filesystems. Only one filesystem per FSP may be fscked. If multiple fsck's for one FSP are issued by the operator, they will be queued by the FSP for sequential execution. The standard fsck command is used to initiate these FSP-based fsck's:

```
piaget# fsck /dev/raxvp/fsp0vp0
** fsp0vp0
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Synchronous Log
** Phase 6 - Check Cyl groups
Files: used 1, avail 33554432
Blocks: used 201, avail 35555455
Fragmentation: blocks 4444431, frags 7, Fragmentation % 0

***** FILE SYSTEM WAS MODIFIED *****
```

There are two types of fsck's for LFS filesystems, fast fsck's and full fsck's.

Fast Fsck

Fast fsck's utilize the journaling functionality of the HTFS filesystem to check only those files which have been recently modified; files which are not present in the intent log can be assumed to be stable. Fast fsck is the default mode for fsck.

Executing fsck

fscks on LFS filesystems are initiated with the standard fsck command

Fast fscks (utilizing the intent log) are the default

Full fscks are done if the intent log has been invalidated or if the -of option is specified

If multiple fsck jobs are issued to a single FSP, the FSP will queue them for sequential execution

Executing fsck

Full Fsck

Performing a full fsck checks the filesystem inode by inode, ignoring the intent log. Note that if the intent log has been invalidated, as occurs when a filesystem is degraded, a fsck will automatically be a full fsck. Full fsck's are initiated with the -of option.

Mounting LFS Filesystems

Filesystems to be NFS shared should always be mounted as LFS filesystems

It is no longer necessary to mount a filesystem to a specified file processor, as the system automatically mounts the filesystem on the FSP where the logical volume is attached

Filesystems to be mounted at boot-time are specified in /usr/AXbase/etc/lfstab, which has a format identical to the Solaris vfstab

Mounting LFS Filesystems

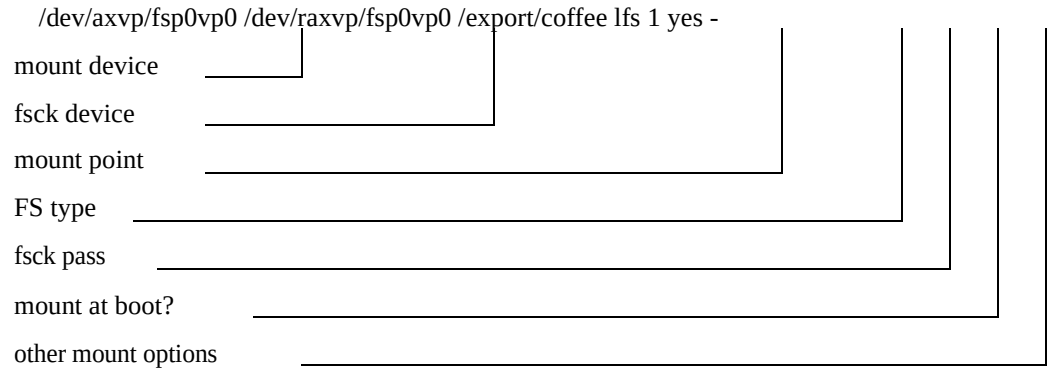
Mounting LFS filesystems

Filesystems intended to be exported for NFS should always be mounted as LFS filesystems, rather than UFS filesystems mounted on the host processor. Specifying LFS causes the filesystems to be mounted on an FSP, so that NFS operations are performed exclusively on the FSP and NP components. While exporting host-mounted UFS filesystems is supported (note that at the time of this writing there are some problems exporting host filesystems through NP network interfaces; bugs are open on the problem), it defeats the purpose of the Auspex hardware and software.

Note: As the functionality of the NetServer File Processor and Storage Processor has been merged into a single File-Storage Processor (FSP), the concept of mapping filesystems to a particular processor is no longer valid. A filesystem will always be managed by the FSP which is attached to the disks the filesystem is stored on.

Specifying filesystems in lfstab

The file /usr/AXbase/etc/lfstab contains a list of filesystems which should be mounted at boot-time. It's format is identical to the standard Solaris vfstab:



- ▲ mount device: the block (cooked) device containing the filesystem
- ▲ fsck device: the character (raw) device specified for filesystem consistency checks (fscks)
- ▲ mount point: directory entry where the filesystem is mounted

LFS Mount Options

rw or **ro**: Specifies whether the filesystem is mounted read/write (default) or read-only

suid or **nosuid**: nosuid will cause the setuid flag to be ignored during execution (default is to respect suid)

nolog: Turns off filesystem journaling; this option offers no significant performance improvement and will result in longer fsck times.

nochkpt: Turns of filesystem checkpointing, which may slightly improve overall system performance, but may not be appropriate for filesystems characterized by bursty data. Note that this checkpointing is unrelated to checkpointing for the purposes of freezing a filesystem for backups or revision control (snapshots).

tmp: This flag is not useful in the context of NFS

fastsync: Improves performance of synchronous write operations by increasing the size of the metadata cache. As with **nockhpt**, the benefits of this option depend on the nature of the filesystem; in this case it may reduce performance on filesystems with very large files.

LFS Mount Options

- ▲ FS type: always lfs in the lfstab file
- ▲ fsck pass: determines when the filesystem is automatically checked
- ▲ mount at boot: specify yes to have the filesystem mounted automatically at boot time
- ▲ mount options: options specified to optimize filesystem performance, see below:

Mount Options

- ▲ **rw** or **ro**: The filesystem is mounted read/write by default; specifying ro will mount the filesystem read-only.
- ▲ **suid** or **nosuid**: Setuid is honored by default; nosuid will cause the setuid flag to be ignored during execution.
- ▲ **nolog**: Turns off filesystem journaling; this option offers no significant performance improvement and will result in longer fsck times.
- ▲ **nochkpt**: Turns of filesystem checkpointing, which is used to reduce the chances that a fsck will be needed after an unexpected system reboot; this option may slightly improve overall system performance, but situations have been seen where bursty data will suffer due to timing problems (see Performance module). Note that this checkpointing is unrelated to checkpointing for the purposes of freezing a filesystem for backups or revision control (snapshots).
- ▲ **tmp**: This flag is not useful in the context of NFS; the gains which arise from its use are masked by network latency, and it introduces the risk of data loss during an unplanned system shutdown. In short, temporary space should always be local, not NFS mounted.
- ▲ **fastsync:** Improves performance of synchronous write operations by increasing the size of the metadata cache. While this can greatly benefit filesystems which see a large number of metadata operations (getattr, setattr, etc.), filesystems which house larger files are better served by a larger user data cache. See the Performance module for more discussion.

Manually Mounting LFS Filesystems

ax_fsck_mount will mount filesystems specified in lfstab, fsck'ing the filesystem first if it is dirty

If mounting with a mount command on the command line, specify "-f LFS"

FSP-attached disks cannot be mounted as standard UFS filesystems

Manually Mounting LFS Filesystems

Mounting filesystems manually

ax_fsck_mount will mount filesystems specified in lfstab, fsck'ing them beforehand if the filesystem is dirty. Filesystems which are already mounted will not be affected by this command.

To mount an LFS filesystem not specified in lfstab, it is neccesary to use the -f LFS option of the mount command. For example:

```
piaget# mount -F lfs /dev/axvp/fsp0vp0 /export/coffee
piaget# mount
/ on /dev/dsk/c0t3d0s0 read/write/setuid/largefiles on Mon Nov 23 14:42:24 1998
/usr on /dev/dsk/c0t3d0s6 read/write/setuid/largefiles on Mon Nov 23 14:42:24 1998
/proc on /proc read/write/setuid on Mon Nov 23 14:42:24 1998
/dev/fd on fd read/write/setuid on Mon Nov 23 14:42:24 1998
/var on /dev/dsk/c0t3d0s3 read/write/setuid/largefiles on Mon Nov 23 14:42:24 1998
/opt on /dev/dsk/c0t3d0s5 setuid/read/write/largefiles on Mon Nov 23 14:46:56 1998
/usr/openwin on /dev/dsk/c0t3d0s1 setuid/read/write/largefiles on Mon Nov 23 14:46:56 1998
/tmp on swap read/write on Mon Nov 23 14:46:56 1998
/export/coffee on /dev/axvp/fsp0vp0 read/write on Mon Nov 23 14:59:05 1998
```

Note that filesystems on FSP-attached disks cannot be mounted as standard UFS filesystems, and attempting to do so will result in an error.

Sharing NFS Filesystems

The share command is used to export NFS filesystems

/etc/dfs/dfstab contains share commands to be run at boot-time

The shareall command will run all the share commands in /etc/dfs/dfstab

Options may be specified to restrict access (see next slide)

Access Lists

Several options accept an access-list argument which restricts the option to specified hosts

The access list is a colon-separated list containing hostnames and netgroups

If DNS is being used for name resolution, all hostnames must be fully qualified, including those in netgroups

A minus (-) sign preceeding a hostname or netgroup will deny the access specified by the associated option

Sharing NFS Filesystems

Sharing NFS Filesystems

Sharing filesystems with the share command

Solaris uses the share command to export mounted filesystems to clients on the network. This is analogous to the exportfs command of SunOS on the current NetServer product. The syntax of the share command is:

```
share [-F fstype] [-o options] [-d
description] pathname
```

- fstype: this must be nfs for all filesystems in the current release
- options: this is a comma separated list which is used to set access permissions to the filesystem as a whole (not to be confused with per file UNIX permissions)
  rw - Makes filesystem read/write accessible for all; this is the default if no options are specified.
  rw=access-list - Allows read/write access to clients specified in the access-list; all others cannot access filesystem at all.
  ro - Makes filesystem read-only for all clients.
  ro=access-list - Allows read-only access to clients specified in the access-list; all others cannot access filesystem at all.
  root=access-list - Allows only the root users of the hosts specified in the access list. Root accesses on all other hosts are mapped to the anonymous user ID (detailed below). By default, no hosts have root access. Netgroups may be used in the access list if the filesystem is being shared with UNIX authentication (AUTH_SYS, see below).
  anon=uid - Sets **uid** to be the effective user ID of unknown users. This defaults to user ID UID_NOBODY (which is set to the nobody entry in /etc/passwd). If this is set to -1, access is denied to unknown users.
  nosuid - By default, clients are able to set setuid and setgid mode. If this flag is set, the server will silently ignore attempts to set setuid and setgid modes on the shared filesystem.
  sec=mode - Specifies the security mode to be used for the shared filesystem. This defaults to AUTH_SYS, specified as sys. If the **mode** is set to AUTH_NONE (specified as none) and a client also

NFS Access Options

rw or **ro** - read/write (default) or read-only access for all clients

rw=access-list - Allows read/write access to clients specified in the access-list; all others cannot access filesystem at all

ro=access-list - Allows read-only access to clients specified in the access-list; all others cannot access filesystem at all

root=access-list - Allows only the root users of the hosts specified in the access list; by default, no hosts have root access

anon=uid - Sets **uid** to be the effective user ID of unknown users (defaults to user ID UID_NOBODY, defined by the nobody entry in /etc/passwd; set to -1, access is denied to unknown users)

nosuid - The server will silently ignore attempts to set setuid and setgid modes on the shared filesystem (default is to allow setuid and setgid)

sec=mode - Specifies the security mode to be used for the shared filesystem (defaults to AUTH_SYS, specified as sys)

Slidetitle

uses AUTH_NONE, each NFS request from that client will be treated as unauthenticated, resulting in the anon value specified above to be utilized. Note that if the client uses a security mode which is not specified for the filesystem, that clients NFS requests will also be treated as unauthenticated. Note also that neither Diffie-Hellman (dh) nor Kerberos (krb4) are supported.

- access-list: This is a colon-separated list which follows an equals-sign (=) in some of the options. It may contain any number of hostnames and netgroups. If DNS hostname resolution is specified in nsswitch.conf, all hostnames must be fully qualified (including those in a netgroup). A minus sign (-) preceding a component will deny access to that component. Please see the share_nfs manpage for details on using this option.
  Note that neither DNS suffixes nor network names are supported in access lists.
- description: Clients will see this description when they request a list of shared filesystems from the server
- pathname: Specifies path to be shared; files in this directory and below will be shared

To share /export/coffee read/write to everyone, issue the following command:

```
share -F nfs -o rw -d "caffiene" /export/coffee
```

Specifying shared filesystems in dfstab

To automatically share filesystems at boot time, share commands should be placed in /etc/dfs/dfstab. The share commands in dfstab have the same format as when they are specified on the command line. Here is a sample dfstab file:

```
# home dirs: only engineering can access
share -F nfs -o rw=engineering -d "home dirs" /export/home2
# library documents: read only for everyone
share -F nfs -o ro -d "library" /export/library
# shared storage: read/write for all
share -F nfs -d "shared storage" /export/storage
```

To share all filesystems specified in the dfstab, invoke the shareall command.

Student notes