NFS FILE SETS

Mike Eisler

mre@eng.sun.com

SunSoft, Inc.

Theresa Lingutla-Raj

traj@eng.sun.com

SunSoft, Inc.

Rob Thurlow

thurlow@eng.sun.com

SunSoft, Inc.

CONNECTATHON ‘97

CONTENTS

- Overview
- Goals
- Partitions and File Trees
- Name Space
- Fileset Locking
- Administration Utilities
- mount re-architecture
- caching changes

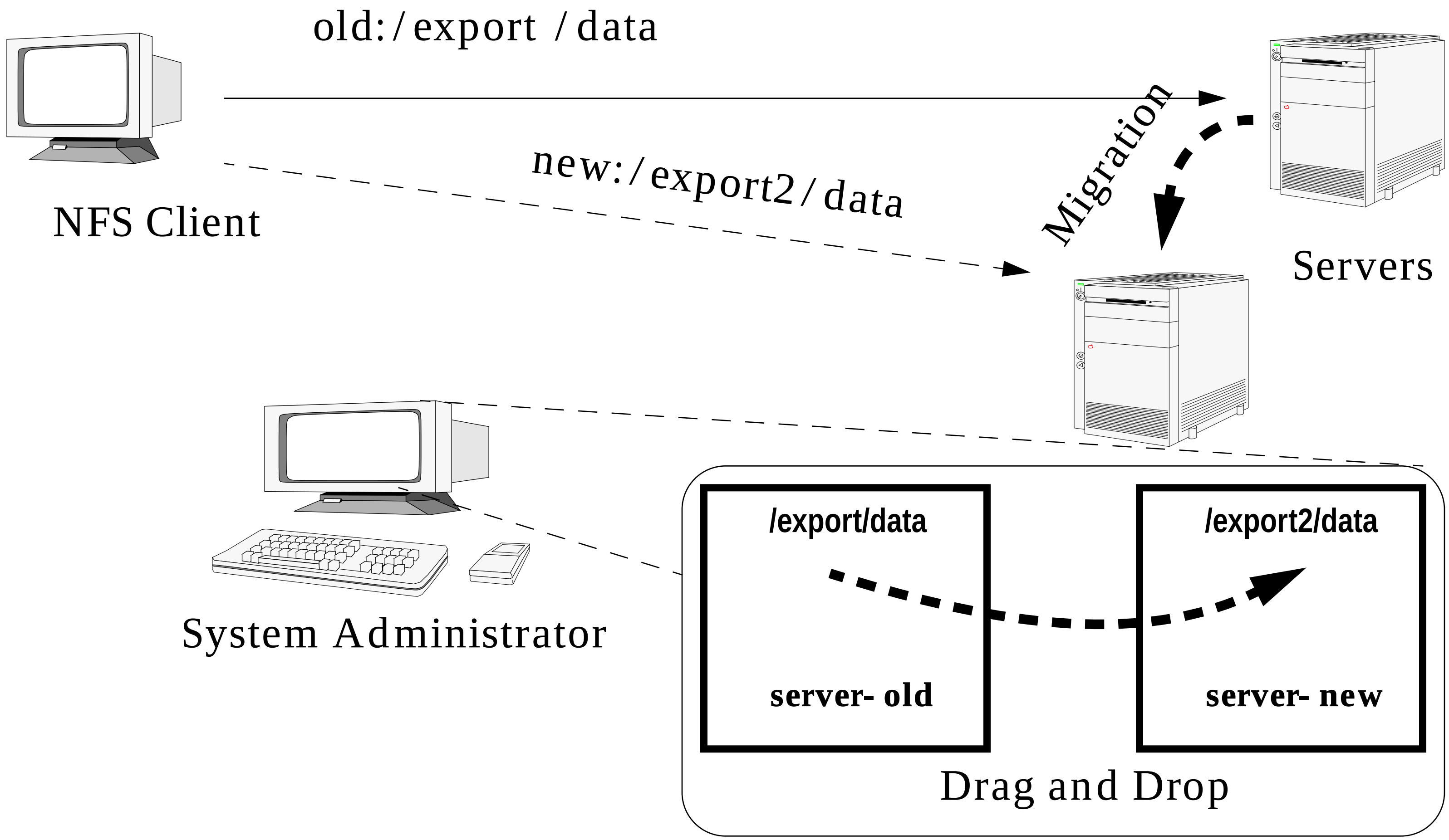
OVERVIEW

- **A file set is a collection of files mounted as a file system by an NFS client**
- **Things to do with file sets:**
  - Migrate them
  - Replicate them
- **Client-side fail over is a basic form of file sets**
  - The NFS client switches to another server by changing IP addresses and re-mapping path names to vnodes
  - Limits:
    - **Read-only**
    - **No consistent file handle, so operations done by other clients can break file identification:**

```
other_client% mv old new
```

- **File set migration is useful in environments where the locations of networked file systems change a lot (or would if NFS had file set migration)**
- **Counter-arguments for file set migration**
  - How often do file systems move really?
    - **ANSWER: not often, but when they do, it is a pain to update name services, maps, and also make the users reboot desktops (or get a new URL)**
  - Why not use clusters?
    - **ANSWER: Clusters are homogeneous**
    - **ANSWER: Cluster nodes must be “near” to each other**
- **File set migration is one of the reasons why people prefer AFS or DCE/DFS (security is the other)**

VISION: LOCATION INDEPENDENT ADMINISTRATION



Disclaimer: the above vision won’t be there in the first phase of NFS File Sets

GOALS

- **No major re-architecture**
  - local file system
  - NFS file system
  - Virtual File System switch
  - existing NFS protocols
- **Ease of administration**
- **Work with prevalent naming systems**
  - Don’t invent naming systems
- **Make disk caching and file sets cooperate**

NFS FILE SETS:

- Partitions
- Transportable File Handles
- Virtual Partitions
- File Trees

Mike Eisler

Sunsoft, Inc.

mre@eng.sun.com

PARTITIONS

- `server# mount -F ufs /dev/dsk/0 /export/home0`
- `client# mount -F nfs server:/export/home0 /mnt`
- **/dev/dsk/0 contains an inode #, each with a generation #**
- **file handles are basically inode#/gen# pairs**
- **Trivial migration:**
    - copy partition (/dev/dsk/0) to another server
    - change IP address in client’s mount table to new server

TRANSPORTABLE FILE HANDLES

fsid issue

- **file handle is typically:**

fsid	**inode# of file**	**generation# of file**	**inode# of exported directory**	**generation# of exported directory**

- **fsid is typically a UNIX device number + a file system type**
  - Even between homogenous systems, the device number is hard to maintain
- **Solution: make fsid a random (64 bit) quantity**
  - near zero chance of collision
  - random number generator uses techniques of RFC XXXX

PARTITION ON DEMAND ISSUE

- **What if the server receiving the partition has no free slice?**
- **Solution: Virtual Partition**
- **Creating a virtual partition on new server:**
  - create a regular file as large as the incoming partition
  - associate the regular file with a slot in a new pseudo device driver
  - newfs /dev/pseudo_dsk/slot#
  - mount -F ufs /dev/pseudo_dsk/slot#
  - Credit for pseudo device driver: Tom Van Baak @Pyramid Technology
  - A poor man’s vnode stack

VIRTUAL PARTITIONS VS FILE TREES

- **File Tree: an arbitrary subdirectory of an exported directory, which can contain files and more subdirectories**
- **Advantage of Virtual Partitions**
  - No more file identify problems
  - Compartmentalizes: simple way to do quotas
  - Better NFS security
- **Disadvantages of Virtual Partition**
  - Lots of partitions complicates administration
  - Fragments space quickly
  - File system within file means twice as much code in data path

- **Advantage of File Trees**
  - Non-invasive to NFS server
  - Potential for exporting NFS mounted file systems
- **Disadvantage of File Trees**
  - requires that path names be recorded for possible re-mapping
- **Will support both forms:**
  - Virtual partitions good for write-shared file systems, such as source trees under source control
  - File trees good for (mostly) single-writer file systems, such as home directories
  - Sometimes migration will go from homogeneous to heterogeneous systems

NFS FILE SETS:

- Name Space Resynchronization
- Locking
- Utilities

Theresa Lingutla-Raj

Sunsoft, Inc.

traj@eng.sun.com

NAME SPACE RESYNCHRONIZATION

- **Client detects fileset unavailability after repeatedly receiving stale filehandle or jukebox error**
- **Client attempts to obtain new coordinates of the fileset**
  - **Automounter**
    - client makes upcall to automounter for latest map information
    - automounter consults naming service
  - **Redirector**
    - nis updates take time to propagate
    - redirector is needed to cover for time lag in name service updates

NAME SPACE RESYNCHRONIZATION

Redirector

- **establish link between old and new location of fileset**
  - **resides on the server in location symbolically represented as “REDIRECTOR_LOCATION”**
    - much like dot-dot symbol implying parent directory
    - avoids assumptions of the pathname supported by the server
    - e.g. /redirector for Solaris
  - **is a symlink**
  - **connected to old location by filehandle**
  - **connected to new location by symlink contents**
    - nfs URL
    - e.g. nfs://newserver//newpath
- **used in conjunction with naming service**

NAME SPACE RESYNCHRONIZATION

Redirector

- **example**
  - filehandle of exported fileset 0x1234

```
client -------- lookup REDIRECTOR_LOCATION ----> server
client <------- fh for /redirector ------------------------------ server

client -------- directory /redirector, lookup 0x1234 ---> server
client <-------- fh for /redirector/0x1234 ------------------ server

client --------- readlink /redirect/0x1234 -----------------> server
client <------- nfs://newserver//newpath ---------------- server

client uses automounter to obtain network address of newserver
```

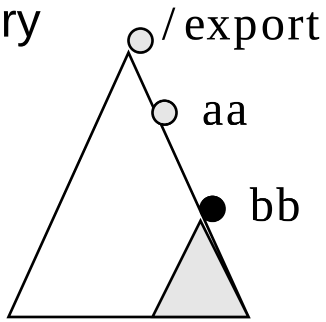
FILESET LOCKING

- **Why?**
  - maintain integrity during transfer
- **Requirements**
  - ease of administration
  - efficient locking
  - efficient checking
- **Design premise**
  - examine lock state on root of mounted fileset
  - extend filehandle to have mounted fileset information
- **filehandle format**

file system identifier	file identifier of the target file	file identifier of the target’s exported directory	file identifier of the target’s mounted directory

FILESET LOCKING

- **types of lock**
  - read/write lock
  - write lock
- **levels of lock**
  - direct lock
  - indirect lock
- **locking a target fileset**
  - set a direct lock on the root of the fileset
  - set an indirect lock on the nodes intervening the root of the fileset and the exported directory



exported directory /export
locked directory /export/aa/bb

○ indicates indirect lock

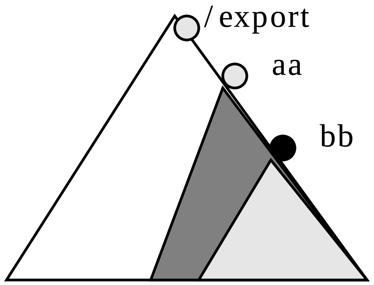
● indicates direct lock

FILESET LOCKING

- **lock testing on every nfs request**
  - **examine mounted directory for lock**
    - if direct or indirect lock then return NFS3ERR_JUKEBOX
  - **examine the exported directory for lock**
    - if direct lock return NFS3ERR_JUKEBOX
    - if indirect lock examine nodes between mounted directory and exported directory. If a node with direct lock is reached then return NFS3ERR_JUKEBOX

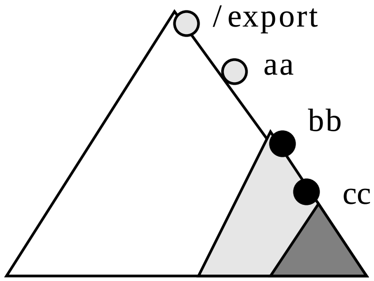
scenario 1 (+ve test)

exported directory /export
locked directory /export/aa/bb
mounted directory /export/aa



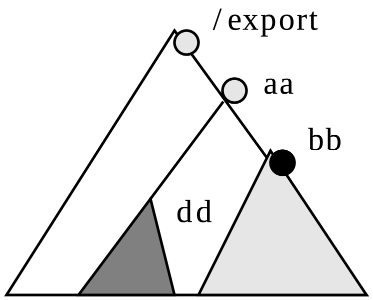
scenario 2 (+ve test)

exported directory /export
locked directory /export/aa/bb
mounted directory /export/aa/bb/cc



scenario 3 (-ve test)

exported directory /export
locked directory /export/aa/bb
mounted directory /export/aa/dd



FILESET UTILITIES

- **fileset creation tool**
  - create subdirectory for pathname method or virtual partition for consistent fh method
  - publish fileset in the name space
- **fileset deletion tool**
  - remove fileset and reclaim space on source server
- **fileset transmission tool**
  - lock the fileset on source server
  - transfer fileset using rdist
  - update name space and setup the redirector on source server
  - unshare the fileset on the source server
- **fileset reception tool**
  - setup virtual partition and UFS filesystem on target server
  - share fileset

NFS FILE SETS:

- Mount Re-architecture
- CacheFS work
- Lock migration

Rob Thurlow

SunSoft, Inc.

thurlow@eng.sun.com

MOUNT REWORK

SVr4 mount code awkward to build on

- **/usr/sbin/mount -> /usr/lib/fs/$FSTYPE/mount**
  - fork()/exec() API obvious but inefficient
  - Poor results for any MT application
  - Cachefs interposition takes a large performance hit and loses control
  - Fileset project needs more direct control of mount(2) args
    - manual vs. automount to support upcall and interposition
- **automountd duplicates much NFS-specific code**
  - Duplicated code for performance and control (e.g. pingnfs())
  - Have to do new work (e.g. failover) and bug fixes in two **very** different frameworks
  - Need MT-hot repackaging of code in one place for automountd

MOUNT REWORK

Solution: shared libraries

- **Shared library with fs-specific mount knowledge**
  - Split code into new logical groupings:
    - premount: collect necessary information (e.g. root filehandle)
    - postmount: e.g. reflect mount in /etc/mnttab
    - preunmount: e.g. is server alive?
    - postunmount: e.g. /etc/mnttab again
  - Filesystems will ship a shared lib containing these entry points
  - These routines will handle hierarchical mounts (e.g. cachefs/nfs)
  - Programs will call find_entry_point() with $FSTYPE and the type of routine needed to get to fs-specific functions
  - find_entry_point() will do dlopen()/dlsym() if not already cached
  - mount_hierarchy() provided for simplicity
  - automountd will keep several dynamic segments mapped for speed

MOUNT REWORK

Shared libraries, cont’d

- **Shared library with generic mount knowledge (e.g. /etc/mnttab update)**
  - Programs will link against this normally
  - Static objects will have a private archive to link against
  - MT-safe: all /etc/mnttab accesses through this library
- **All Solaris filesystems will use common code**
  - It’s not just a good idea, it’s the law
  - fork()/exec() support for 3rd party filesystems will be maintained

MOUNT REWORK

Shared libraries, cont’d

- **Code example**

```
int (*premount)(int, char *[],int, struct mountargs **);
int (*postmount)(int, struct mountargs *);

main(int argc, char *argc[])
{
        premount = find_entry_point(“ufs”, PREMOUNT);
        postmount = find_entry_point(“ufs”, POSTMOUNT);

        err = mount_hierarchy(argc, argv, premount,
                postmount);
        exit(err);
}
```

CACHEFS REWORK

Plumbing changes

- **CacheFS keeps a persistent data/metadata cache**
  - Stores server name as part of CacheFS organizational data
    - /var/cache/jurassic:_export_home6_thurlow:_home_thurlow
  - Stores file handle for each cached vnode
  - We need a way to update this metadata when fileset migration occurs
  - This is the same reason failover and CacheFS didn’t play in 2.6
- **Solution: cachefsd**
  - Kernel will upcall to cachefsd to provide new information
  - cachefsd will update this itself or via a private system call
- **Will look for other opportunities to make CacheFS integrate better with NFS**

LOCK MIGRATION

- **When fileset moves, what happens to locks?**
  - Ideal result: client negotiates locks with new server and then drops lock held on old server
    - Would like a grace period on fileset on new server, but likely can't get that without protocol change
    - Could run into problems with contention from other clients
  - Non-ideal result: client waits for locks to be dropped before permitting a particular rnode to be rebound to new server
    - Requires that old server keeps providing service for awhile, which may be reasonable
    - Can a client do I/O under these circumstances?
  - Worst-case result: send SIGLOST to processes which hold a lock on a migrating fileset
    - Yuck! Breaks transparency pretty badly
  - Possible to represent lock state in future server-to-server protocol
  - Still in the early stages of thinking about this