The Restore-o-Mounter

The File Motel Revisited

Joe Moran
Bob Lyon
Legato Systems, Incorporated

Abstract

We present a scheme for referencing and accessing saved[1] files in a manner that is transparent to UNIX® applications. The scheme requires no kernel modifications. Instead, it uses a "mounted" process that allows users to change directories to the past and browse their saved files with their favorite utilities. The mounted process acts as a protocol gateway between NFS™ and a commercially available network backup product. Time travel is supported; users may change directories to any moment in the past. Any saved version (not just the most recent version) of any file can be viewed or recovered, even if the file has since been deleted.

Using this transparent method of retrieving saved files by naming their location in the past, a poor man's file migration scheme can be implemented by substituting a symbolic link to a saved location for a file. Once a file is referenced, the symbolic link can be replaced with its original file. This migration scheme requires no kernel modifications yet remains transparent to UNIX applications and users.

Introduction

This paper describes two file management features[2] that have eluded UNIX users - file recovery integrated into the operating environment and transparent file archiving (more commonly called file migration). Since saved files can be named and accessed with the same ABI as normal files, users can employ their favorite UNIX tools to find and restore deleted or damaged files. The most obvious examples of such tools are **cd, ls**, **find** and **cp**. Equally important though are the users' environments - shells like **csh** or **ksh** that are customized to each individual and provide services such as file name completion and pattern matching, or even windowing environments that completely hide the normal UNIX commands.

File archiving and migration strategies are driven by economics and convenience. Most users do not mind that "inactive" files are removed from their local disks and archived elsewhere provided that the files are easily and rapidly recoverable when needed. Migration of this sort is economical because low capacity, high performance and high priced disks contain only frequently accessed files, while high capacity, low performance and low cost media (like optical disks or tapes) contain seldom or never accessed files. Users find it far more convenient to archive files to free local disk space than to perform the numerous tasks and endure the long elapsed times associated with ordering and installing new and bigger disks.

Customers might also wish to "actively" archive complete directory sub-trees after important events occur at their companies. For example, when a software company ships a major release of its product, it might want to immediately archive that release including documentation, SCCS files, optimized and debuggable objects for numerous platforms and perhaps even the compilers and system libraries used to build the release. A second example is a company that archives a departed employee's home directory, thereby freeing much needed disk space without risking the deletion of important data.

---

1. We use the word "save" to denote the super set of "backup" and "archive"; save is also easier to conjugate than backup.
2. The features described in this document should not be construed as products features currently available from Legato.

These features are of very limited use if they are not available in the networked environment.

Why delivering solutions is hard

Ease of use is the key attribute to any solution involving file recovery. In the UNIX environment, this usually means the solutions have to be compatible with and transparent to users' existing tools. Since the common ABI for all tools is the UNIX system call interface, this generally means that a solution must be implemented as enhancements to the kernel's filesystem primitives. The Virtual File System [KLEI86], VFS architecture was introduced as part of the implementation of Network Filesystem [SAND85] to separate the filesystem interface from the various filesystem implementations. VFS clearly made adding filesystem functionality easier provided that the implementor had access to the kernel source and understood the rules associated with extending the kernel. However, independent software vendors (ISVs) cannot depend on the VFS interfaces; they are not identical on each UNIX vendor's platform and each new release of a specific UNIX variant can render the ISVs' added functionality useless until the code can be re-ported to the new kernel.

Hardware costs also conspired to keep solutions unavailable. The price of media jukeboxes allowed only the richest and most dedicated companies to consider deploying automated recovery systems. The limited size of this niche market further discouraged UNIX ISVs from producing software that assisted in the automation. However, within the last twelve months, the price of tape jukeboxes has plummeted while the reliability of their robotics has risen sharply. For example, one can buy a 50 gigabyte (before compression) tape jukebox with tape drive for less than $5,000. With compression and larger jukeboxes, end-user cost might drop as low as three cents per megabyte.

About NetWorker

Legato sells a backup and recovery product called NetWorker [LEGA93]. NetWorker is designed and implemented using a client/server architecture. A server is a machine that manages the saved media (usually tapes or optical disks) and an "on-line index" consisting of two databases. The first database maps a file's key and a time to the file's attributes. The second database maps a file's key and a time to a location in the server's media pool where the complete file's attributes and data are saved. Note that the NetWorker server knows no details about its clients or the clients' files and filesystems. The server is required to retrieve file attributes (quickly) and file data (a little more slowly) when and if a client demands them. The server supports most known tape devices, optical disks, and normal files. Many popular media robots are also supported for "unattended" operation.

The NetWorker client is the machine with files and filesystems that need to be saved. The client walks the local filesystems and sends their descriptions and data to the server. In the case of a UNIX client, the files' keys are both file names and device-inumber pairs, while the files' attributes are identical to NFS attributes. A UNIX directory has no data, but its attributes include a list of all files found within the directory. The client knows nothing about how its files' attributes and data are archived by the server; it only knows that server can deliver either on demand.

The product's client side includes file browsers that allows a user to view saved filesystems as of any time in the past. The command line interface to the browser is similar to BSD **restore -i** with additional features for time specification and file version information. The product also includes X Window System™ browsers for command-line challenged users. The browser translates user inputs into queries against the server's on-line index and presents the results as a view into a saved filesystem. Once the user has selected files for recovery, the browser submits a request to the server for data associated with the desired files.

Clients communicate with servers via an application specific protocol built on ONC™ (Sun) Remote Procedure Calls [NOWI88]. The name of the application protocol is "NetWorker Save and Recover" or NSR. Typical protocol stacks are NSR/RPC/TCP/IP or NSR/RPC/SPX/IPX. NetWorker server platforms include eight UNIX vender's platforms as well as NetWare®. Client-only implementations are also available for DOS and numerous other UNIX platforms.

More details of NetWorker's design and implementation are covered in NetWorker's theory of operations manual and beyond the scope of this paper. Most of hard design issues were addressed years ago during NetWorker's devel-

opment, and the features of the restore-o-mounter are built upon these mature solutions.

Ib: the file index browser

When you lose a file, why can't you just "change directory" to the past and copy the file back to the present? This simple question was the inspiration for the restore-o-mounter. The first reason why you can't do that is because the past is not a mounted filesystem and normal users can't use **mount** anyway. However, the automounter [CALL89] mounts filesystems for normal users on demand. The automounter is a UNIX process that mounts itself as an NFS server in the UNIX filesystem. The automounter then turns name references sent from its kernel into mounts to actual filesystems. The automounter also periodically attempts to auto-unmount the filesystems that it mounted.

The index browser (**ib**) takes a similar approach. By default, it mounts itself on the mount point **/ib**, but instead of translating NFS requests from its kernel into mount actions, the requests are translated into browse sessions with NSR servers. Figure 1 shows the conceptual workings of the index browser; it is essential a protocol gateway that translates from the NFS protocol to the NSR protocol. When presented with a new client name or a new time to browse the saved files, **ib** forks and execs a copy of **iba**, the index browser agent. The generic form of names that **ib** translates to **iba** sessions is

`[[`*`server`*`:]`*`client`*`][@`*`time`*`]`

Server is the name of the NSR server to use; note that NSR client software usually determines the appropriate server. The ability to specify the NSR server is needed only in environments where a single NSR client has multiple NSR servers. *Client* is the name of the NSR client whose saved files will be browsed and potentially recovered; the default client name is the local machine's name. *Time* determines which view will be browsed; the default time is the most recently saved version. **Ib**'s most visible function is to choose suitable and intuitive defaults and convert human usable time strings into NSR suitable time values. **Ib** uses **getdate** routines [BELL87] to provide user friendly time translation. Here are some examples of changing the working directory into some root directories in the past:

```
cd /ib/@now
```

changes directory to the most recent save for the local machine.

```
cd /ib/@yesterday
```

changes directory to yesterday's save for the local machine.

```
cd /ib/ganymede
```

changes directory to the most recent save for the machine ganymede.

```
cd /ib/jupiter:io
```

changes directory to the most recent save for the machine io saved to the NSR server jupiter.

```
cd /ib/jupiter:io@last_month
```

changes directory to the save of a month ago for the machine io saved to the NSR server jupiter.

By the time the **ib** process receives an NFS lookup or stat on a name in **/ib**, the kernel has already concluded that the name was not a filesystem mount point. Therefore, **ib** creates a new directory whose name is derived from the name

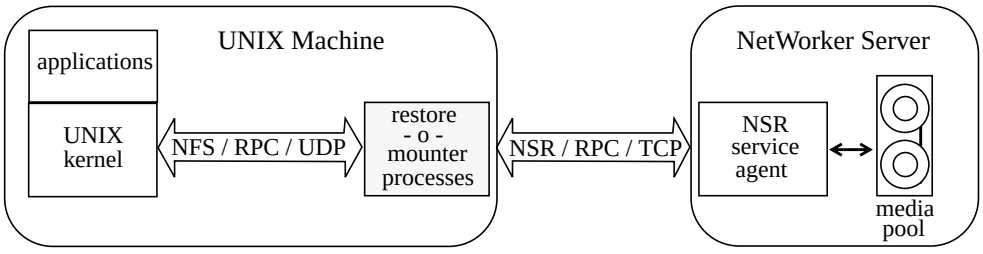

Figure 1. The restore-o-mounter forks processes that translate the NFS protocol originating from the local kernel to the NSR protocol destined for a NetWorker ("backup") server. The NetWorker server is not necessarily a UNIX machine or even an NFS server.

submitted by the kernel, mounts an index browser agent on the derived directory, then creates the submitted name as a symbolic link to the derived directory, and finally returns the symbolic link information to the kernel. For example, after a user references a file from "yesterday's" save of the local machine, **/ib** may look like:

```
io% cd /ib/@yesterday
io% ls -l /ib
total 2
lrwxrwxrwx  1 root 10   Apr 9 10:33 @yesterday -> jupiter:io@04-08-93_10:33:46
drwxr-xr-x 25 root 1536 Apr 6 09:02 jupiter:io@04-08-93_10:33:46
```

One can see that the local machine's name is io and its default NSR server is jupiter. The **getdate** routine mapped "yesterday" to April 8, 1993 at 10:33:46 am, which is exactly twenty four hours prior to the execution of this example.

Ib periodically attempts to unmount the mounts that it automatically performed. Upon a successful unmount, the mount point and symbolic link are removed. We found that removing the mount point caused problems with programs that remember the true mount point path names (e.g., the OpenWindows™ File Manager and **ksh**). To alleviate this, **ib**'s time translator appends or removes an extra underscore character, `'_'` at the end of the time string when the time string is identical to the string submitted by the kernel. So, the contents of **/ib** may eventually look like:

```
io% ls -l /ib
total 2
lrwxrwxrwx 1  root 28   Apr 9 11:27 jupiter:io@04-08-93_10:33:46 ->
jupiter:io@04-08-93_10:33:46_
drwxr-xr-x 25 root 1536 Apr 6 09:02 jupiter:io@04-08-93_10:33:46_
```

should the unmounted name be re-submitted.

Iba: the index browser agent

Figure 2 shows the results of **ib** forking and execing two index browser agent (**iba**) processes. Like **ib**, **iba** is a "mounted" NFS server process that only responds to calls from its kernel. When launched by **ib**, **iba** mounts itself on top of a directory fabricated by **ib**. **Iba** establishes an NSR connection subject to the arguments passed to it by its par-

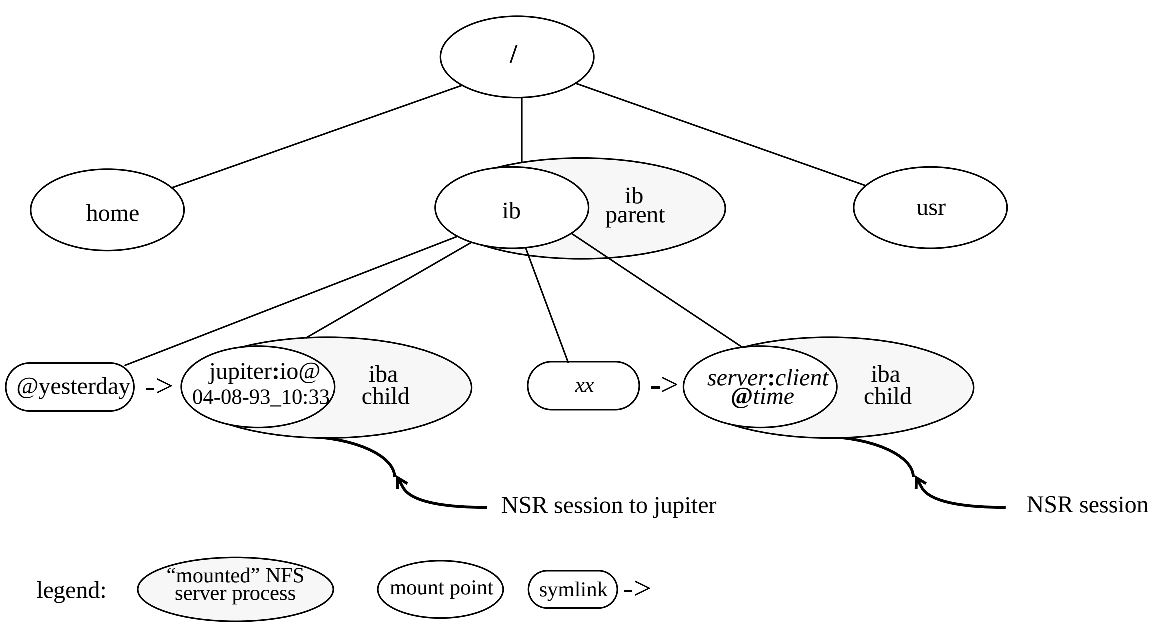

Figure 2. The typical mount points in a UNIX filesystem after two browsing sessions are initiated with the NetWorker server.

ent. The arguments include which NSR server to bind to, which NSR client's index to browse, which view (in time) to present to its kernel, and where to mount itself in the UNIX file name space.

As mentioned before, NetWorker's on-line index for a UNIX file contains the complete stat information of that file. If the file is a symbolic link, then the link's value is also kept on-line; this means that NSR media need not be accessed to resolve symbolic links. Because NFS and NSR file attributes are similar and because the NFS and NSR architectures are similar (since the original designers of both architectures have considerable overlap) **iba**'s primary task is implemented by one-to-one mapping of NFS operations to NSR operations already available in NetWorker's browsers through a library. The challenge in most NFS server implementations is the design of the NFS file handle (fhandle). **Iba** fhandles contain actual pointers to cached in-core data structures built by the library as files are referenced. I**ba** fhandles also contain information used to verify their validity. The size and speed of **iba** are covered in a later section.

Iba servicing NFS reads

Iba has command line arguments that determine one of three policies for servicing NFS read requests:

1. Always attempt file recovery on NFS read.
2. Do file recover on NFS read if data can be recovered automatically without requiring any human intervention. This is the default. NetWorker software knows if the volume(s) that contain the requested data are mounted on a device or available in a robotic jukebox. If so, the data is considered "near-line" and **iba** will fetch it. If the data is not near-line, **iba** returns ERREMOTE[3] as the results of the NFS read operation.
3. Never attempt file recovery on NFS read. The results of the NFS read operation is always ERREMOTE. This is handy if the customer does not want the NetWorker server bogged down doing actual data recovers and desires a "stat-only" filesystem. In this case, the restore-o-mounter could more appropriately be called the "browse-o-mounter".

The latter options are not found in related work but is useful in the restore-o-mounter, because the NetWorker server most likely stores the data on tape. The seek performance of this class of media is at least three orders of magnitude

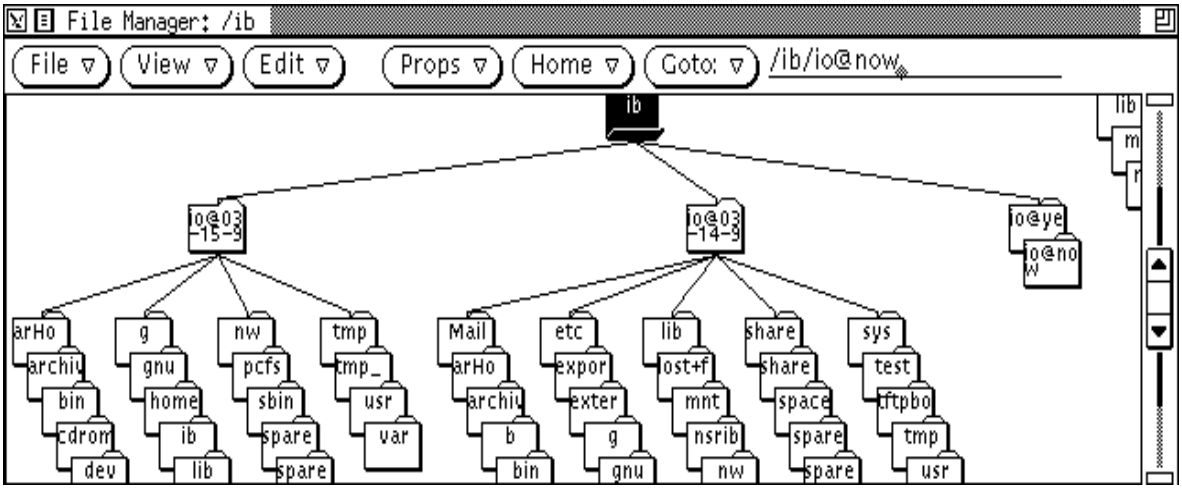

Figure 3. A typical application's view of restore-o-mounted filesystems. The two sub-trees were generated by the user pointing the OpenWindows File Manager to the directories `/ib/io@now` *and* `/ib/io@yesterday`. *In this example, numerous directories were deleted between yesterday and today. Note that the two folders at the far right are symbolic links to their corresponding sub-trees.*

3. ERREMOTE is not an error defined in the NFS protocol specification. It is defined as part the System V RFS™ definition. Its associated error string "Object is remote" almost matches the desired "the data is too far away". This is probably good enough for a system that was wont to print "Not a typewriter". Clearly **iba** is relying on its NFS client, the local kernel, to pass most errors up to its callers without any attempt to interpret them.

slower than that of rotating media. So, while fetching a few files from tape via **iba** may be practical, **grep**'ing one's entire home directory in the past for a pattern is not recommended. NetWorker's recover command is more suitable for moving vast amounts of data from tapes to disks. Still, users agree that a wealth of information is available from stat-only filesystems.

Once a read request is performed, **iba** requests the entire file from the NSR server. The file is then cached in a more traditional filesystem on the UNIX machine. (The location of the file cache is yet another command line argument to **iba**.) The original read operation is not responded to until the entire file is recovered. Subsequent reads are then serviced from the cache. File caching is the optimal way of dealing with the very cheap, but very low performance tape media. However, a subsequent section shows how the cache may become the real thing.

The UNIX kernel is multi-threaded, but **iba** (like most user NFS server processes) is singly-threaded. To avoid suspending all lookup service for the duration of any file fetch, **iba** forks a child to handle the recovery of each file. The parent then resumes NFS service but ignores read requests associated with children who are actively recovering the data. That is, a read request is ultimately answered by the parent **iba**, but only after a child has placed a complete file in the cache.

Versions and hidden files

The NetWorker product provides a means to see every saved version of any file. **Iba** employs hidden file techniques to see and explicitly name these versions. A hidden file name never appears in the results the NFS readdir call, but the same name yields successful results when used as arguments to the NFS lookup call.

As originally described in The File Motel [HUME88], versions of a file *f* may be found in the hidden directory

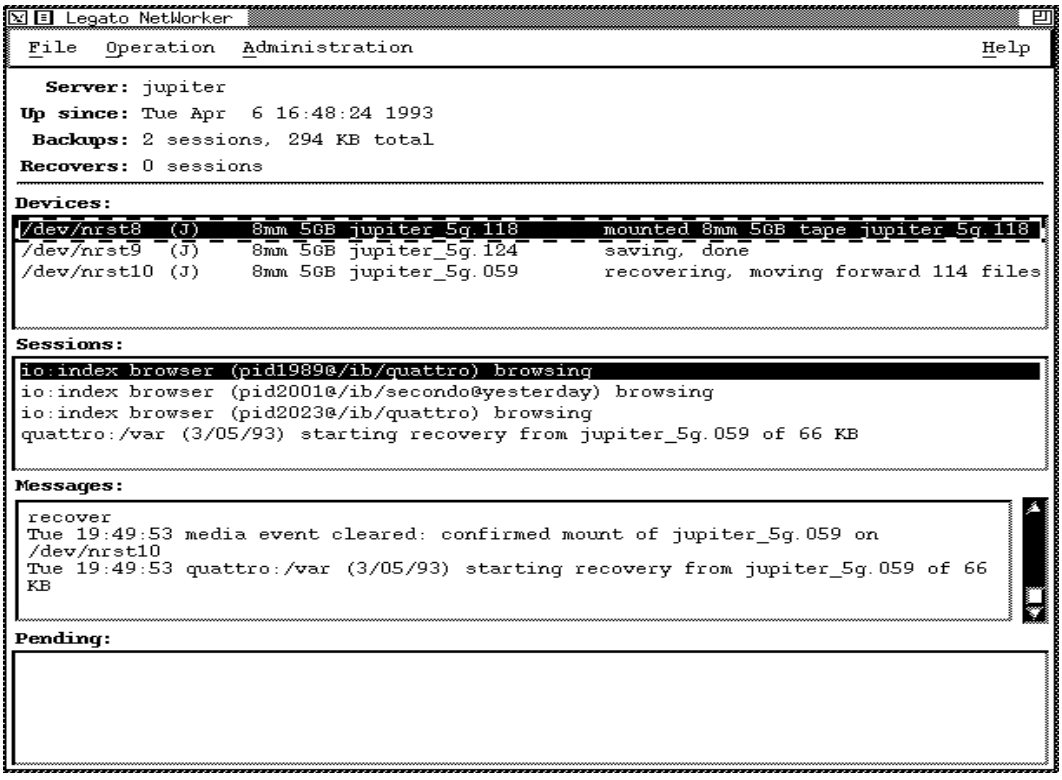

Figure 4. This is a screen shot of a NSR Motif™ based console monitoring the NSR server jupiter. All four sessions originate from two browsing sessions on the machine io. The first browser is navigating the machine quattro's saved files and has forked a child, pid 2023, to recover some of quattro's data (indicated by the last two lines of the Sessions panel). The data may be awhile in coming since 114 tape file marks must first be skipped (indicated by the last line of the Devices panel). The second line of the Sessions panel shows that the browser on io is navigating the machine secondo's filesystems as of yester-

f.**V**, independent of **iba**'s browse time. For example,

```
io% cd /ib/io/etc
io% ls -l rc.local.V
total 28
-rw-r--r-- 1 root 7564 Apr 1 09:03 v1:_jupiter_5g.124_at_\dev\nrst9
-rw-r--r-- 1 root 7396 Mar 5 08:39 v2:_jupiter_5g.113_at_Engn_Jukebox
-rw-r--r-- 1 root 7361 Jan 9 19:58 v3:_jupiter_5g.062_at_Engn_Jukebox
-rw-r--r-- 1 root 7096 Dec 3 12:32 v4:_jupiter_5g.018
```

shows all the saves of the machine io's **/etc/rc.local** file. The generic name of files in a versions directory is

```
v#:_volume[_at_location]
```

where **v1** through **v***n* are assigned to versions, beginning with the most recent save. *Volume* names the save media containing that version. *Location* is provided if it is known. In this example, the most recent save of **rc.local** is on a tape that is currently mounted in **/dev/nrst9**. In order to have legal file names, **iba** translates any forward slashes in the location information into back slashes. The next two versions reside in the engineering jukebox, while the last version on the tape "jupiter_5g.018" may be sitting in the non-automated portion of the media pool. The listed modifications times above indicate that the file was saved because it changed. Files are also saved during "fulls"; listings of their versions are pretty boring. For example,

```
io% ls -l /ib/ganymede/vmunix.V
total 6401
-rwxr-xr-x 1 root 828881 Jan 11 09:17 v1:_jupiter_5g.124_at_\dev\nrst9
-rwxr-xr-x 1 root 828881 Jan 11 09:17 v2:_jupiter_5g.114_at_Engn_Jukebox
-rwxr-xr-x 1 root 828881 Jan 11 09:17 v3:_jupiter_5g.101_at_Engn_Jukebox
-rwxr-xr-x 1 root 828881 Jan 11 09:17 v4:_jupiter_5g.063_at_Engn_Jukebox
```

shows that the **vmunix** built for ganymede has not changed since its installation. **Iba** usurps all the access time values and replaces them with the (NSR supplied) time when the save occurred. So by using the **-u** option to **ls**, we see

```
io% ls -lu /ib/ganymede/vmunix.V
total 6401
-rwxr-xr-x 1 root 828881 Apr  2 22:25 v1:_jupiter_5g.124_at_\dev\nrst9
-rwxr-xr-x 1 root 828881 Mar  5 22:41 v2:_jupiter_5g.114_at_Engn_Jukebox
-rwxr-xr-x 1 root 828881 Feb  5 21:49 v3:_jupiter_5g.101_at_Engn_Jukebox
-rwxr-xr-x 1 root 828881 Jan 11 21:38 v4:_jupiter_5g.063_at_Engn_Jukebox
```

that ganymede's **vmunix** was saved soon after it was built and late in the evenings of the first Friday of each month thereafter.

Users should approach hidden directories with caution since **pwd** can't find their names:

```
io% cd /ib/ganymede/vmunix.V
io% pwd
pwd: getwd: read error in ..
io% cd ..
```

Iba also supports naming non-directory files at any point in time with the simple syntax *f***@***time*. These names are also hidden, and they remain independent from **iba**'s browse time. For example, the following two commands are identical:

```
io% strings vmunix.V/v3*
io% strings vmunix@Feb_6
```

Yclept files

The previous section dealt with **iba** hiding versions of files in various directories. A different type of hidden file is one that is hidden by time. This corresponds to the case where users knows the name of a lost file, but does not recollect a date (usually in the distant past) when they last had the file. The users could eventually locate their file by sys-

tematically traveling backwards in time until they find the date when the file last existed, but this could be cumbersome.

Iba provides another feature inherited from NetWorker to avoid the tedious searching in time. If a user can name a file, he can recover it. **Iba** requires no special syntax for naming files (including directories) obscured by time. However, the exact name is required since the name does not exist in the current version of its parent directory. Once yclept, files remain legitimate (not hidden) members of their parent directories.

Final words on ib and iba

The file handles provided by **iba** may contain pointers to data structures (nodes) associated with referenced files. These nodes are currently never freed, so the persistence of file handles is guaranteed. Two obvious consequences are that mapping a file handle to its associated data is trivial, but in the worst case may require large amounts of virtual memory. The average measured size of **iba**'s file node is 115 bytes, including malloc overhead. The storage is mostly used by the NFS stat structure; actual file names are a distant second. Because file access through **iba** is casual and because **iba** exits once auto-unmounted, anticipated swap requirements when caching all file nodes is still quite reasonable. Should **iba** memory usage become a problem, an alternative implementation of **iba** can use less caching and place more of a burden on the NSR server by regenerating **iba** file nodes on demand from other information contained in the **iba** fhandle.

The designs of the NFS file handles are different in **ib** and **iba** because these two programs have very different job descriptions, though both are NFS servers. **Iba** is a application level gateway between the NFS and NSR protocols. **Ib** provides ease-of-use features by auto-mounting and launching **iba** with the appropriate arguments. The original deployment of NFS separated launch (mount) from operation and it has since proven to be a good idea [PUGS84]. We anticipate other methods for launching the existing **iba** at points spread throughout the traditional UNIX filesystems.

Performance.

The **find** command was used to traverse a machine's filesystems in three ways - local UFS, via NFS from across a quiescent Ethernet™, and via the restore-o-mounter to an NSR server across the same quiescent Ethernet. In all cases, the machines involved are Sun SPARCstation-2™ with 64Mb of main memory and 3.25 inch SCSI disks with 13.5 msec. average access time.

The NSR server managed 416 gigabytes from 34,000 distinct save sessions from 56 clients. The client chosen for the benchmark had 398,000 file instances in NSR server's on-line index. Each of the client's filesystems had at least three full saves (stretching back at least three months) in the on-line index.

Access method	Files found	Seconds elapsed	Normalized UFS	Normalized NFS
UFS, pass 1	71441	621	1.00	0.61
UFS, pass 2	71452	750	1.00	0.60
NFS, pass 1	71284	1120	1.64	1.00
NFS, pass 2	71286	1144	1.67	1.00
IB, pass 1	70979	1890	2.76	1.67
IB, pass 2	70979	479	0.70	0.42

Figure 5. The relative times needed to run the find command against filesystems accessed via UFS, NFS, and the restore-o-mounter's index browser processes.

The identical **find** command is run twice to show the caching effects of UFS, NFS and the restore-o-mounter. Figure 5 shows the results. Note that NFS takes 64% - 67% more time than local UFS. In the first pass, the restore-o-mounter takes 67% more time than NFS, or 176% more time than UFS. Most of the restore-o-mounter time for the first pass is attributable to the NSR server, and not to the **iba** process. The second time the command is run yields little difference in the performance of UFS and NFS. But the restore-o-mounter blazes away, taking only 42% of the time that NFS takes and only 70%[4] of the time that UFS takes!

Each pass of the **find** benchmark shows the extremes of the relative lookup and stat performance of the restore-o-mounter. Neither are very realistic given the usage model of the restore-o-mounter. Read performance was not benchmarked because it is highly dependent on the speed of the underlying media holding the file data. Obviously, minutes may pass before any data is forthcoming from a tape jukebox. Related work asserts that these long lag times are reason enough to exclude tape media from any serious consideration of transparent file recovery services. Our experience does not bear this out; users tolerate seemingly long delays because the process is entirely automated, highly reliable, and provides feedback (via the NetWorker monitors) regarding its progress. Once users trust the system, they usually switch to another task (take a coffee break or read e-mail) and switch back after their files are recovered.

Transparent file migration

We have shown how the restore-o-mounter provides application transparent file access to saved files. In this section we present a new application that exploits restore-o-mounter features to provide a poor man's file migration utility.

Note that NSR clients distinguish backup data from archive data. The NSR server separates these two types of data into different media pools. The media with backup data is eventually deleted or recycled according to customer policies; after all, the data is merely a copy of the real thing.

Archived data is expected to live forever. Rather than being a copy, it is assumed that the data is the real thing. As the NetWorker product ships today, files are archived from explicit user actions. Once the files are archived, users may then choose to explicitly remove[5] some files. In doing so, it is up to the users to remember their actions and to recover the files should they ever be needed.

The best way to remember removed files is by systematically making notes about them in the filesystem. Related work incorporates new information into a file's inode. This has the advantage that users or applications need not perform any special actions to recover removed files when running a modified kernel that automatically reacts to references to the new type of inode.

Our scheme also makes notes about removed files. But rather than extending the inode information, a removed file is simply replaced by a symbolic link to the restore-o-mounter name of the archived file. For example, a user may record all his outgoing e-mail messages in a file that he periodically moves to his notion of a mail archive. If the full path name of such a file was **/home/io/mojo/mail.record.92**, then the associated symbolic link might appear as:

```
mail.record.92 -> /ib/jupiter:io/home/io/mojo/mail.record.92@03-03-93_11:52:47
```

Note that the resolution of the symbolic link contains the explicit save time of the archived file. Omitting the time would cause an infinite loop once the symbolic link itself is saved and the symbolic link within the **iba** filesystem is dereferenced! The following symbolic link is functionally equivalent:

```
mail.record.92 -> /ib/jupiter:io@03-03-93_11:52:47/home/io/mojo/mail.record.92
```

But, while the first link can use any existing **iba** session for the client io to the NSR server jupiter, the second requires an explicit **iba** session at the time 03-03-93_11:52:47. Therefore, the first style of link is used to avoid unnecessary process forking.

---

4. The biggest advantage that the restore-o-mounter has over the traditional filesystems is that its files are static and its metadata can be cached in a small amount of virtual memory. Never-the-less, we hope this astounding number helps to debunk threes myths: code runs faster in the kernel; marshalling data through XDR is inefficient; context switching to user level (file) services is expensive.

5. In some of our competitors products, file removal is automatic. This is euphemistically called "filesystem grooming".

Two nice features fall out from the symbolic link approach to migrated files. The first is that user may rename their files without losing the associated archives. Second, the symbolic links are saved during traditional backups. So if an entire directory or entire disk is lost, the traditional recovery replaces the symbolic links; all the archived data is not placed back onto the disk.

Unmigration

As mentioned before, **iba** usually caches a complete file's data when it processes an NFS read request. The restore-o-mounter can be invoked with an option which recovers files back to their original locations when the files' data is accessed. The following conditions must be met if **iba** is to recover a file in place:

- the file name presented to **iba** must be of the form *f***@***date* where *date* is exact match of *f*'s save time
- the file cannot overflow the filesystem's space
- the original location of the file must still be a symbolic link that directly resolves to the file name presented to **iba**

The last rule implies that renamed migrated files cannot be recovered in place. Heuristics could be added to **iba** to allow it to hunt down the renamed symbolic link and recover to it. A simple heuristic would be to look for the symbolic link in the original parent directory. A more complicated one could be to have the NetWorker **save** command build a special symbolic link cache as it performs its daily backups. (If the file is recovered from tape, then **iba** may have quite a bit of clock time to burn; searching for the renamed symbolic link could be accomplished then.)

Use of the **-L** option to **ls** (use **stat** instead of **lstat**) hides the fact that a file is migrated:

```
io% cd /home/io/mojo
io% ls -lLt mail.record*
-rw------- 1 mojo  985484 Apr 10 23:42 mail.record
-rw------- 1 mojo 4871761 Jan  3  1993 mail.record.92
-rw------- 1 mojo 6720668 Jan  2  1992 mail.record.91
```

while no **-L** option to **ls** indicates the files' real status:

```
io% ls -lt mail.record*
-rw------- 1 mojo 985484 Apr 10 23:42 mail.record
lrwxrwxrwx 1 mojo     57 Mar  3 11:54 mail.record.92 -> /ib/jupiter:io/home/io/
mojo/mail.record.92@03-03-93_11:52:47
lrwxrwxrwx 1 mojo     57 Mar  3 11:54 mail.record.91 -> /ib/jupiter:io/home/io/
mojo/mail.record.91@03-03-93_11:52:47
```

Here we use **egrep** to recover in place a file from a tape jukebox, and use **ls** to see its new status:

```
io% /bin/time egrep presto mail.record.92 > /tmp/foo
       253.8 real       5.7 user       2.1 sys
io% ls -lt mail.record*
-rw------- 1 mojo  985484 Apr 10 23:42 mail.record
-rw------- 1 mojo 4871761 Jan  3  1993 mail.record.92
lrwxrwxrwx 1 mojo      57 Mar  3 11:54 mail.record.91 -> /ib/jupiter:io/home/io/
mojo/mail.record.91@03-03-93_11:52:47
```

The filesystem that **iba** exports for recover in place is mounted for read and write access since a file could be modified by applications. The applications may continue to access the file via its NFS handle into the **iba** process while subsequent opens of the file will be handled by the file's originating filesystem. Thus one can modify a currently migrated file and have the correct behavior (e.g., **cat** >> *currently_migrated_file*) .

The final observation is that in place recovery can be performed by a restore-o-mounter running on any machine. The example above was performed on the machine io which was also the originator of the migrated files. The example could have been performed on ganymede which NFS mounts **io:/home/io**. This demonstrates why both the NSR server and NSR client are named in every symbolic link.

Migration

Readers may have resigned themselves to the fact that this paper only addresses recovers and not saves. Despair no more. We allocate the following half page to describing archive saves.

The **migrator** was implemented by modifying the NetWorker **save** command to declare its saves as archive data. Then normal save policies were replaced with migration policies. Policies include:

- the file's type
- the file's change time
- the file's size
- the file's owner and permissions
- the number of exact copies of the file on archive media

Only regular files are automatically migrated. Files greater than 16 Kbytes were typically chosen. Some systems also provide upper bounds for candidate files. Filtering files owned by special users or with certain permissions is also a good idea. For example if we migrate an executable owned by root, then we may end up migrating the kernel, shared libraries, single-user utilities like **fsck**, and even **iba** itself.

Some users feel that the file should be archived on multiple media before it is replaced with a symbolic link. The **migrator** checks this by reading the versions directory of a candidate file as it is presented by **iba**. If there are enough duplicate versions located on different volumes, then the **migrator** replaces the file with a symbolic link to the last such version. If a symbolic link is not substituted for the file but the file is a candidate for later migration, the **migrator** then saves the file to archive media. To actually migrate a file, the **migrator** must visit the file at least twice. The first time it saves the file to archive media; the last time, it replaces the file with the corresponding symbolic link.

Caveats

The **migrator** is an application that we expect to traverse the filesystems once a day, once a week, or maybe only once a month. A daemon that polls for the free space in filesystems could be implemented to launch a **migrator** should the free space drop below some threshold. Without kernel support, the **migrator** cannot protect applications from ENOSPC errors caused by short term anomalous behavior.

Recovery in place of archived files by NFS clients can only succeed if the NFS clients shares a common view of mounted filesystems with the NFS server.

Re-migrating a recovered in place but unmodified file may occur; this is not desirable. The unnecessary re-migration could be avoided if **iba** could control a restored file's change time.

Related work

Several earlier systems have provided a filesystem interface with time travel capability (primarily to access versions from past backups).

The restore-o-mounter was originally inspired by The File Motel [HUME88] which implemented a Version 8 File System in addition to its own access commands to browse and recover past backups to optical disks. The restore-o-mounter provides additional functionality by extending the name space to allow specification of the NSR server, NSR client, and browse time. It also allows specification of an arbitrary version of any file within the restore-o-mounter filesystem and includes in place recovery of archived files.

The 3DFS™ [ROOM92] uses NFS access to a filesystem constructed from previous backups to an optical disk jukebox. 3DFS has many similarities to the restore-o-mounter and NetWorker, but has a number of differences in the design and implementation.

- Both 3DFS and the restore-o-mounter look like a filesystem using a standard NFS interface. Both 3DFS and the

restore-o-mounter allow you to use unmodified UNIX commands to browse and access old versions of files. Both accept dates for files at any point in the filesystem hierarchy. While the restore-o-mounter restricts the dates for a directory tree to the top of the tree (i.e., a mount point), 3DFS allows time travel at arbitrary points in the tree. But, the greater flexibility 3DFS may present a less consistent view of the filesystem name space.

- 3DFS uses only optical disk jukeboxes. The restore-o-mounter (via NetWorker) can use a variety of save media (e.g., 8mm and 4mm tapes, optical disk) with or without jukebox support.
- 3DFS uses remote NFS mounts to a dedicated server, while the restore-o-mounter uses only local NFS mounts to a gateway process. Using local NFS mounts makes it easy to figure out when a mount is no longer busy (the unmount system call doesn't return EBUSY), provides control of NFS access (e.g., **ib** sets up reasonable timeouts and retransmission parameters), and makes it possible to provide better file migration support (it can recover files in place). The disadvantage of using local NFS gateway processes is that there can be no centralized network-wide caching of recovered files (although this functionality can be provided by the NetWorker server).
- 3DFS has more complicated naming rules and **glob**'ing rules for increased flexibility, but sometimes requires special commands. Each directory can have a date associated with it, but this date is not visible via the standard UNIX **pwd** command. The restore-o-mounter has simpler file naming conventions and doesn't require any special commands to access version information, but provides less flexibility for accessing the data. We view the less complicated naming conventions and fewer options as a benefit.
- 3DFS internal access methods are strictly path name based, so it does not handle file or directory rename situations gracefully. For example, renaming a directory renames all the files under it and breaks their history. The restore-o-mounter (via NetWorker) uses both path name and file id access methods and properly handles rename situations. This allows a correct view of a filesystem as of any time (subject to the times of when saves are performed).

The Plan 9 filesystem [PIKE90] uses a true filesystem on optical disk and a two-level cache to provide transparent filesystem access to previously backed up files. It also provides automatic file migration. However, Plan 9 provides support only for files in the Plan 9 filesystem while the restore-o-mounter works with any traditional UNIX filesystems.

The Inversion filesystem [OLSO93] uses the POSTGRES database system to provide fine grained time travel and transactional operations. However, to take advantage of these facilities applications currently need to be rewritten to use new programmatic interfaces provided by a special library that must be linked with each application.

A number of systems provide a more tightly integrated file migration solution by providing custom OS's or modifying the kernel. The BUMP project used kernel modifications to provide automatic file migration hooks [MUUS89]. Epoch and NetStor provide file migration products based on NFS servers. Epoch has traditionally used specialized file servers to provide automatic file migration services. Currently Epoch is moving away from specialized kernels and is working to define UNIX kernel hooks to allow for user processes to handle the bulk of the file migration services [WEBB93]. The restore-o-mounter file migration philosophy differs from these approaches in that it requires absolutely no OS changes at the cost of some functionality.

Conclusions

We have shown that both transparent file recovery and transparent file migration can be accomplished with no modifications to the kernel. We avoided the VFS abstraction because an ISV cannot take advantage of it when delivering products into the ever changing UNIX markets. A far more stable and universal interface, the NFS protocol was readily combined with our NSR protocol to provide the new functionality. Symbolic links, often reviled by users, were exploited as stubs for migrated files, making our scheme transparent to any application and compatible with any backup method.

The Automounter philosophy is useful for moving functionality out of the kernel and providing new privileges for normal users. We believe that extending the UNIX system via new processes will soon become the norm and not the exception.

Finally, the architecture described in this paper allows a variety of devices (not just optical jukeboxes) to be practical for both backup and file migration.

Acknowledgments

Joe Moran implemented all aspects of the restore-o-mounter after Bob Lyon said it should be easy. Bob Lyon wrote most of this paper. The restore-o-mounter builds upon the NetWorker product which is implemented and deployed by the team at Legato Systems.

Dave Cohrs, Bill Nowicki and Linda Weinert helped to turn our random thoughts into mostly coherent English.

We give special acknowledgment to Tom Lyon for envisioning and championing the mechanisms that allow NFS servers to do more than just share files.

References

[BELL87] Steven M. Bellovin is the author of **getdate** routines, public domain software written in 1987 acquired from the University of North Carolina at Chapel Hill.

[CALL89] Brent Callaghan and Tom Lyon, *The Automounter*, 1989 Winter (San Diego) Usenix Conference Proceedings.

[HUME88] Andrew Hume, *The File Motel - An Incremental Backup System for Unix*, 1988 Summer (San Francisco) Usenix Conference Proceedings.

[KLEI86] Steven R. Kleiman, *Vnodes: An Architecture for Multiple File System Types in Sun UNIX*, 1986 Summer (Atlanta) Usenix Conference Proceedings.

[LEGA93] Legato Systems, *NetWorker Product Overview*, 1993, Palo Alto, CA.

[MUUS89] Michael John Muuss, Terry Slattery, and Donald F. Merritt, *BUMP - The BRL/USNA Migration Project*, November 1989 LISA Conference Proceedings, Monterey, CA.

[NOWI88] William I. Nowicki, *RPC: Remote Procedure Call Protocol Specification*, RFC 1057, Network Information Center, USC ISI, Marina del Rey, CA, 1988.

[OLSO93] Michael A. Olson, *The Design and Implementation of the Inversion File System*, 1993 Winter (San Diego) Usenix Conference Proceedings.

[PIKE90] Rob Pike, Dave Presotto, Ken Thompson, and Howard Trickey, *Plan 9 From Bell Labs*, 1990 Summer UKUUG Conference Proceedings, London.

[PUGS84] Tom Lyon, *Why separate mounts from the NFS service*, 1984, private communication to Sun's Network File System's group.

[ROOM92] William D. Roome, *The 3DFS: A Time-Oriented File Server*, 1992 Winter (San Francisco) Usenix Conference Proceedings.

[SAND85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon, *Design and Implementation of the Sun Network Filesystem*, 1985 Summer (Portland) Usenix Conference Proceedings.

[WEBB93] Neil Webber, *Operating System Support for Portable Filesystem Extensions*, 1993 Winter (San Diego) Usenix Conference Proceedings.

Trademarks

this paper. 3DFS, Ethernet, Motif, NetWare, NFS, ONC, OpenWindows, RFS, SPARCstation-2, Sun-3, SunOS, UNIX, and X Window System are trademarks of their respective companies.

Biographies

Joseph Moran is a principal engineer at Legato Systems Incorporated which he cofounded in 1988. At Legato, he has had major roles in architecting and implementing all of Legato products including Prestoserve™, the client and server sides for UNIX and NetWare versions of NetWorker, and NetWorker for DOS in addition to pitching in where ever needed. From 1984 to 1988 he was a Member of Technical Staff at Sun Microsystems. While at Sun he worked on a variety of tasks for the SunOS™ kernel including implementing the virtual memory system, doing the original Sun-3™ SunOS port, bringing SunOS up on a number of new machines, and designing and implementing **kadb** after getting tired of debugging kernels on bare hardware. From 1982 to 1984 he was a System Programmer at Hewlett Packard where he was involved with various UNIX related activities including working on the original BSD UNIX port to the Hewlett Packard Precision Architecture. He received a M.S. in Computer Science in 1982 and a B.S. in Electrical and Computer Engineering in 1980, both from the University of Wisconsin, Madison. He can be reached via e-mail at **mojo@Legato.COM**.

Robert B. Lyon is the Vice President of Core Technologies at Legato Systems Incorporated which he cofounded in 1988. At Legato, he has exerted architectural influence on the implementation and deployment of the company's products, and at times, implemented components like NetWorker's database for its on-line file indices. From 1983 to 1988 he was the Project Leader and Manager of Sun Microsystems Network File System group where he played a similar role after implementing Sun's Remote Procedure Call and eXternal Data Representation (RPC/XDR) package. From 1979 to 1983 he was a Member of Technical Staff at Xerox Corp Systems Development Division where he assisted in the implementation and tuning of all levels if the XNS protocol stack and had project lead responsibilities for the Clearinghouse Name Service. Prior to Xerox, he was an MTS at Bell Labs in Holmdel NJ where he had system administration responsibilities for a small lab of UNIX machines; he claims to be the first to deploy **uucp** outside of Murray Hill. He received a M.S. in Electrical Engineering from Stanford University in 1978 and a B.S. in Engineering from Cornell University in 1977. He can be reached via e-mail at **blyon@Legato.COM**.