Moving Large Filesystems On-Line, Including Exiting HSM Filesystems

Vincent Cordrey, Doug Freyburger, Jordan Schwartz, and Liza Weissler – Collective Technologies

ABSTRACT

Since the advent of Logical Volume Managers [LVM], larger individual disk drives, and high uptime expectations, it is no longer possible at some sites to schedule downtime windows long enough to move some very large or very critical filesystems to new hardware. Hierarchical Storage Management [HSM] systems share this problem. While at some sites, users continue to enjoy the functionality of HSM based on their specific usage patterns, sites whose usage patterns do not match HSM's strengths have a more acute case of the same problem when moving their data to new hardware. This paper presents a unique approach to moving filesystems that *permits a system to remain on-line and accessible.* New terminology is also introduced to assist discussion: **Forward Relocation**, **Reverse Relocation**, and **Hybrid Relocation** are defined and basic algorithms are presented. While it is true that the total throughput rate of a traditional dump and restore is higher, the methods presented here require nearly zero downtime.

The authors have used these techniques to relocate data on filesystems with many small files during the working day at several sites, as well as to exit HSM systems as part of standard technology refresh programs. Three case studies, where both types of data were relocated, are described in their basic detail as successful (and ongoing) implementations. The authors know of no prior works on this topic and hope to foster further discussion and refinement of the techniques.

Introduction

In plotting the rehosting of large filesystems long outage windows seem inevitable. With the ideas presented here, the authors hope to expand the rehosting debate by potentially reducing gaps in data availability traditionally associated with a relocation of data from one server to another or even within the same server. While not a perfect solution for all filesystem relocation projects, some installations may find the nearly continuous data availability attractive enough to consider implementing one of these techniques.

Background and Terms

Uptime Expectations

As the total user population increased, computing pervaded the corporate desktop and email is now a mission critical service. Thus, users have come to expect production uptime from fileservers. These users are not interested in computers for their own sake; rather, they use their systems as tools, so they are less tolerant of outages. The standard has gone from users who viewed their computers as sports cars that they expected to tinker with, to users who view their computers as telephones with screens. This leads to the expectation of a "dial tone" whenever they reach for their keyboards.

So Many Files, So Little Time . . . For Now

Users accumulate very large numbers of files because they hoard their data and email messages for *years.* Nearly all methods for copying filesystems take much longer with large numbers of files than with fewer files taking up the same amount of total storage. The longer copy times can mean that it may take an entire shift to copy a large filesystem (even when a full weekend is prepended to the outage window) and users will not tolerate that much downtime.

Logical Volume Managers [LVM] aggregate many disks into very large filesystems. Also, some systems use automounters to supply home directories to their users from many filesystems. In either case, the users now demand high uptimes, and long scheduled outages are not acceptable at some sites. Thus, the ability to relocate filesystems **On-Line** during the day shift is important, especially at some sites that are short-handed in support personnel.

As faster filesystems and networks are designed, this trend will reverse, but for the moment, the pendulum of changing technology has given rise to filesystems too large to relocate off-line during an acceptable outage window, and has created the need for **On-Line Relocation**.

Terminology

Early work with this subject revealed that, to prevent confusion, we needed to establish some fairly strict terminology. Even this paper required careful corrections of usage.

This subject deals with the movement of files, directories, and indeed, whole filesystems. Whether it is a discussion of HSM, our new techniques, or traditional methods, data and files ebb and flow from one place to another. Thus, we have tried to choose language that permits the type of movement to be differentiated. We have adopted vendor specific terms, industry usage, and attempted to eliminate overloading.

Basic HSM Terms

medium *(pl. media)* – the smallest discrete storage unit addressed as a whole; a tape volume, optical platter, physical disk, logical volume, etc.

migration – the movement of files to deeper levels within HSM Systems; implies the probable return of the file to its original location.

migrated – absent from the medium (level) being examined as a result of migration.

resident – opposite of migrated; object is present on the medium (level) being examined.

stage-in – to recall from a deeper level of HSM and make resident.

stage-out – to migrate to a deeper level of HSM and remove from source medium.

Paper Terms

full on-line – both source and destination filesystems or directories remain in read-write mode.

semi on-line –at least one of the source or destination filesystems or directories is not read-write.

relocation – movement of a file or files from one filesystem or directory structure to another as different from migration; implies a single, permanent movement of the file.

scatter gather – placed and retrieved from non-contiguous locations, different media, or different directories.

systematic – placed and retrieved from contiguous locations, the same medium, or the same directories.

HSM General Principles

Hierarchical Storage Management [HSM] refers to software that permits automatic migration of data from on-line storage, usually magnetic disk, to lower cost secondary and perhaps tertiary storage such as optical disk or tape. UNIX implementations of HSM came to the fore in the late 1980's. HSM helped answer the call for very large storage systems at a time when large capacity spinning disk servers for UNIX systems were expensive and not commonly available outside of the realm of super computers.

HSM systems work very well for some types of data, and many sites continue to enjoy excellent service from them. However, some early HSM adopters now find themselves with aging systems that are at or near the end of their useful life, plotting the relocation of files either to large fileservers, newer brands of HSM, or true archival systems.

HSM systems perform well with a small number of large files but are commonly used at installations where there are a large number of small files. HSM systems have a high storage processing overhead and are therefore inappropriate for small files.

HSM is philosophically different from true archival systems, but it is commonly abused as such. Multiple copies back to a baseline or some limitation on the number of copies is an "oops" recovery feature, not an archive, and an "undelete" feature is not explicitly supplied on many systems. A trash can recovery feature is also not an archive. More to the point, an HSM system is designed to keep accessible nominally one *most recent* copy of a file. Archiving involves keeping a specific copy of a file that represents the state of the file at a specific time. Additionally, an archiving system allows access to a large number of versions of a given file that represent specific versions accumulated over time. While this may sound like a revision control system, archiving can be used even with files that are too large for reasonable differencing, and archiving normally also involves storage on less expensive media that may be near-line or off-line. Good archive systems also index all of the versions of the files that have been archived – they keep track of multiple versions of the same file with the same name.

What Changed?

Several factors have changed the technologies so that large capacity fileservers are now common, and HSM systems are no longer the only choice for storage of large amounts of data:

- Spinning disk media is far less expensive and much larger single disk drives are now available.
- LVMs are now available that allow systems to aggregate many disks together into very large filesystems.
- There are now vendors that specialize in large capacity, high performance fileservers.
- On the user interface front, "drag and drop" GUIs encourage misuse as users copy entire directory trees wholesale from place to place without regard to where the data might be stored.
- The labor costs to maintain HSM systems are high: the systems are multilayered, complex, and require highly skilled labor to keep them running.
- An unanticipated aspect of HSM labor costs is the fact that both media volumes and databases require manual garbage collection. This requires a high degree of skill to accomplish, and without it, performance gradually declines.

Why Relocate in the First Place?

The simple motivation to *relocate* the data rather than *abandon* it is that you want to *keep* it. At sites running large non-HSM fileservers, a constant technology refresh program is required to stay current: additional, larger, faster disks are installed; different filesystem software is added (such as journaled filesystems); and entire servers are replaced. In these cases, data on previous generations of hardware and filesystems must be relocated. This was previously done during a scheduled outage window. Given large amounts amount of data and the time it takes to relocate it, the ability to relocate data on-line during the working day has become extremely useful.

Contemporary with the changes in technology, several reasons to exit HSM systems have surfaced. HSM systems, by their very nature, do not provide real-time access to all of the data. With everything a click away, waiting is no longer acceptable. Diminishing HSM expertise makes the systems hard to maintain in an operational state – *everyone who knew how to deal with it left,, and you're stuck holding the bag.* Backups on some HSM systems can be slow to complete and add extra layers to a backup scheme, as a full backup now only represents data resident on spinning disk. The internal complexity of HSM systems can make them unstable; HSM systems have several failure modes. Some systems suffer from inter-component communications failures which lead to an interruption of service that may require a full system reboot to clear. Media failures plague some installations, while index databases and data files can be corrupted by filled filesystems on some others. Lastly, because the systems were designed in the late 1980s, some of them are not Y2K compliant.

Techniques

The technique used to relocate the data from a filesystem depends on three basic choices:

- File selection (**scatter gather** vs **systematic**)
- Data availability (**Full On-line** through **Off-line**)
- Relocation algorithm choice (**Forward**, **Reverse**, **Hybrid**, or **Just Plain Copy**)

File Selection

File selection algorithms largely do not matter when non-HSM systems are being relocated: the files are usually on the same medium, and there is a very small penalty for selecting them at random. With HSM systems, however, the algorithm used can have a significant impact on the time required to relocate the data. While this section discusses two basic file selection algorithms as they apply to HSM systems, there may be some cases when the concerns addressed here should be applied to non-HSM systems.

A Word About Scatter Gather Versus Systematic

HSM media volumes, be they tape or optical, are created as needed. This means that the mapping between files in an HSM medium and files in a directory appears to be random. Some HSM systems deliberately try to distribute the files onto a larger number of media to limit the impact of losing any one medium by spreading the migrated files across many media. So any given directory or tree will likely have files on many media volumes.

The definition of the Scatter Gather technique is to ignore the underlying HSM architecture and file distribution. The basic strategy of Scatter Gather is **Just Plain Copy**. Variations are to copy the entire system all at once or one chunk at a time, usually by directory. With one chunk at a time, planning must be done to avoid frequent mount table changes. Both of these variations require the system to be healthy and take a very long time, because they churn the system at all of the different HSM levels.

The definition of Systematic for this paper is to *not* cross media boundaries by using a knowledge of the underlying HSM architecture. This implies a layered approach scoped within migration levels. Systematic file selection tends to be much faster because it deliberately controls media mounts. It can also clear filesystem space permitting the stage-in of files at deeper migration levels without triggering new migrations to make space available.

Approaches are tailored to the level being evacuated. Any approach at one migration level may appear Scatter Gather at other levels. The main strength of a Systematic approach is that the HSM system is *not* required to be healthy. Non-healthy systems can have the healthy parts evacuated first and this can improve the health of the system. The Systematic approach can skip over the non-healthy parts of an HSM system, permitting creative [NON-Front-Door] approaches to be used for this ''inaccessible data.''

User Data Availability

Taking the system **Off-Line** is often the first strategy considered. On some non-HSM systems, the outage window required to relocate all of the data is small enough to be acceptable. This is the standard dump-restore paradigm not covered by this paper. Very large, very critical, or HSM systems require too much time to copy to be able to relocate the data in an outage window short enough for their uses or users.

Putting the system in **Read-Only** mode is a **Semi Off-Line** strategy. This prevents new data from being added to the system. However, it does not end migration churning on HSM systems, because users continue to access their own files as a part of their regular usage. It also means a significant change in work process for some sites which use their filesystems as a primary working area.

Leaving both the old and new storage **Read-Write** is a **Full On-Line** strategy and is the primary focus of this paper. New data can be created during the relocation process, and the work process is minimally impacted. In particular, this strategy was used while developers were actively running make in their filesystems. An example of moving a critical filesystem would be relocating /usr [describing the freeing of blocks for running programs is left as an exercise for the reader]. This technique can be used to eliminate downtime, or to turn a long outage window into a quick reboot. On the other end of the spectrum, some

HSM systems can take months to relocate their data which is why a single outage window is unacceptable.

Algorithm Descriptions

Just Plain Copy

The basic mechanism is to use a standard copying tool like cpio, tar or dump, and replace entire directories or sections with symbolic links as each directory or section is completed. This is similar enough to common Systems Administration practice that no pseudo code is presented.

Pros

- "Really Easy"

Cons

- *Very* slow (jukebox trashing, filesystem churning).
- Requires **Read-Only** or **Off-Line** mode (*downtime*).
- Since process is slow:
  - Requires new mount points or remounting of new storage on old mount point.
  - Requires user retraining for changed mount points.
  - Requires user education regarding access to old files.

Forward Relocation

The basic mechanism is the individual replacement of files on the old storage with symbolic links pointing to the new storage after each file has been copied. The authors have used this option on all of the non-HSM filesystems in the case studies.

Pros

- **No downtime**
- New storage doesn't require pre-population
- Don't have to reboot any clients before starting

Cons

- Directory collapses can lead to stale NFS handles
- New files in non-collapsed directories written to old storage
- Hard Linked files left for last
- Special handling required for emacs and mh

Forward Relocation Algorithm Pseudo Code

- *Force Flag*
- *Unconditional Flag*
- emacs *problem*
- mh *problem*

```
find source objects on old storage
    # this may mean a list of files
    # just staged-in on HSM Systems
while ( source )
does source not exist?
    ignore it and get next object
        # pointless to relocate nothing
is source a relocation link?
    check to see if it has been renamed
    (basename of link text != basename)
        # might be emacs, mh or mv...
        if renaming wouldn't overwrite
        existing file on dest,
            rename destination
if source is a directory
    use cpio to duplicate it
    if duplication successful,
        set ownership and permissions
    loop until no further changes:
        renames_needed = 0
        compare all relocation links
        (link text !~ basename)
            renames_needed++
            if wouldn't overwrite
                rename dest.
    end loop
    if renames_needed != 0
        report error:
            link renaming problem
        get next object
    is directory empty
    or only leave behind links?
    yes: if collapse flag is set
        collapse it with rm -rf
        and replace with
        leave behind link,
    no: ignore it
        and get next object
general check for all remaining types
    if target exists and not force flag
        ignore it and get next object
        # prevent overwrites
        # helps with "emacs"
        # and "mh" problem
    if source older than target
    and not unconditional flag
        ignore it and get next object
        # prevent double relocation
if source is a symbolic link
    is it actually safe to relocate it?
        # two pages of discussion
        # and comments in the code
    yes:    relocate it with cpio
        and replace with link
    no:     ignore it
        and get next object
if source is a file
    if ignore migration flag
        is file migrated?
           ignore it
           and get next object
```

```
    if link_count > 1
        if name not in inode table
            record inode num and
            name in inode table
        count instances in inode
        table for inode number
        ( in_table >= link_count )
        yes:    names listed valid?
            copy and replace all
            remove from inode tbl
        no:     get next object
any other file type
    use cpio to duplicate it
    and replace with leave behind link
end while
```

Reverse Relocation

The basic mechanism is to pre-populate the new storage with the tree of *directories* without copying any files and make symbolic links pointing to the *files* on the old storage. While it is not required, transparent access to the new storage can be provided by remounting the old storage on a different mount point and mounting the new storage on the original mount point. Finally, the links are replaced on the new storage with files from the old storage. Because there is a potentially large outage window during the pre-population, the authors have only utilized this method on systems that can be placed in an **Off-Line** or **Read-Only** state. However, following pre-population and client reboots as necessary to remount the new storage, the systems can be returned to the **Full On-line** state.

Pros

- New files written to new storage.
- Access to new storage does *not* go through old storage.
- Handles emacs and mh problem better.

Cons

- Requires pre-population of entire filesystem with directories and symbolic links (*really hard with 500,000+ files to do in one outage window.*)
- Requires outage window or read-only during pre-population with symbolic links.
- Requires Client reboot if new storage is mounted transparently on old mount points.
- Requires User training if new storage is on a new mount point.
- Not transparent to user (extra outage window or read-only).
- Hard Linked files left for last.
- Open files may produce stale NFS handles.

Reverse Relocation Algorithm Pseudo Code

- *Assume primary user reference point is through new storage*
- *Then new storage has true names*

```
duplicate directories only
                on new storage
make symbolic links in new storage
    pointing to files on old storage
find reverse links on new storage
    # this may mean a list of files
    # just staged-in on HSM Systems
while ( rev_link )
if rev_link points to a symbolic link
    is link a relocation leave behind?
    no: if safe,
        pull it over
        using rev_link's name
        and delete it
    yes:    delete it
if rev_link points to a file
    is file migrated?
        if not force copy,
            get next rev_link
    if link_count > 1
        if src_name not in inode table
            record src_inode num,
            src_name, and rev_link
            in inode table
        count instances in inode
        table for src_inode number
        ( in_table >= link_count )
        yes: src_names listed valid?
            rev_links listed valid?
            pull over all
            using rev_link names
            and delete
            remove from inode tbl
        no:     get next rev_link
        # this might leave
        # a few behind, but
        # the inode table
        # shows which ones...
    pull it over
    using rev_link's name
    and delete it
if rev_link points to a directory
    # we really should not see any
    # directories passed to us
    # but we can handle them...
    if error_on_directory
        report an error
        get next rev_link
    is src directory is empty
    or only leave behind links?
    yes:    collapse it with rm -rf
    no:     get next rev_link
any other file type
    pull it over
    using rev_link's name
    and delete it
end while
```

Hybrid Relocation

The basic mechanism is to use **Forward Relocation** for all rapid access media and switch to **Reverse Relocation** for slower media. The Pros and Cons and pseudo code are as described in the two cases above. This is useful for very large HSM systems, or ones that have more than one migration level.

Another **Hybrid** technique uses a modified **Reverse Relocation** which acts more like a **Forward Relocation** in that it leaves behind forward relocation links on the old storage when it replaces the reverse links on the new storage. This technique can be used to permit valid access through both the old and new storage and allows a **Reverse Relocation** to be used without remounting the new storage on the old mount point. However, having both paths available to users can be somewhat problematic and this technique eliminates the inherent emacs and mh compatibility of **Reverse Relocation**. [The code starts to look very much like **Forward Relocation**.]

Pitfalls

Double Relocation Problem

When doing multiple passes over the source filesystems, careful checking must be done to avoid relocating relocation links destroying valid data on the destination storage by creating self referential relocation links. This is especially the case when directories are being deleted and folded into single symbolic links, where it is easy to cross into the new storage without noticing. The actual scripts or programs used must check for this at several points.

Pathological Filename Problem

With files created by PCs, Macs and GUI applications, filenames that contain shell special characters have become common. (In this case, white space is considered a shell special character since it is a field separator.) These can be handled in a number of ways. If there is a small number of such files, find them first and correct their names in place. If there are no ''quote'' characters, try to protect them from being interpreted by the shell. A more general approach is to use STDIO instead of command line parameters to pass all filenames. This last suggestion works for everything except filenames that have embedded carriage returns, newlines or nulls.

emacs and mh

These applications present special challenges because they rename files rather than reusing the inode on the other end of a symbolic link. At some sites, front end interfaces to mh have an additional behavior that makes use of **Forward Relocation** difficult: these front end programs fork and change their working directories to the mh directories. When these directories are collapsed they become symbolic links and the front end programs exit. Thus, in some cases, it is best to do directory collapsing when users are logged off.

If mh only deleted files by renaming them, that would not be too bad, but it then reuses the filenames it has cleared up. The effect is that mh renames its files for most operations and will desynchronize the two filesystems. To keep the filesystems synchronized, an additional step must be taken, and the relocation worker process must not relocate newer versions of the files with the same names.

cpio under SunOS

On SunOS, cpio always returns a zero exit status, so its exit status cannot be used from within scripts or programs. The System V version of cpio does not have this problem.

Post Copy Checksumming

Post copy checksumming using md5 would be a good feature to implement on unreliable networks and for the justifiably paranoid.

Inode Creation Optimizations

For better performance, each symbolic link can be created prior to directory collapse or file copy and mv used to shift it into place following removal. [mv is slightly faster than creating an inode with ln -s.]

End Game

Forward Relocation

When relocations have completed, the source filesystem will have been collapsed down to a single or at most a few symbolic links for top level directories. The final goal is to have the new filesystem mount from the same point as the old filesystem. This is where even **Forward Relocation** may require some client outage.

In an automounted environment where the filesystem is occasionally quiescent on the client (not held open by some process on the client) the automounter can be made to perform the remounting of the new storage on the old mount point transparently. Changing the automount tables (and signaling the automounters on the clients) to have the new storage mounted both on the new or temporary mount point and the original mount point of the old storage will cause the client machines to use the new storage exclusively the next time they access and remount the original reference mount point.

A day or a week later, most of the clients will have ceased using the old storage. Those clients that have not remounted the new storage on the old mount point can be determined by inspection of the old server's showmount output. Direct intervention can sometimes be done on the client: killing the process that has the old storage open and waiting for the automounter to unmount it before restarting the process avoids a client reboot. If this fails, those few clients that remain can be rebooted, but *no server outage is required.* Once the old storage has been unmounted from all clients, it can be taken off-line.

In statically mounted environments, remounting can also be done one client at a time. With a bit of skill and luck, only a few clients will require reboots.

Reverse Relocation

At completion, the new storage will have no reverse links left in it. Any files left on the old storage are probably abandoned (deleted on new storage) or replaced by newer versions on the new storage.

If the new storage was mounted on the old mount point before relocation was started, then the desired appearance has been attained. If a temporary mount point was used for the new storage then some remounting may be necessary. However, if the new mount point can become a permanent reference, then only an unmounting of the old storage is required. If remounts are required, a client outage may need to be scheduled, and user retraining undone (since the new storage used a *temporary* mount point).

In an automounted environment, the automount tables can be changed and client automounters signaled. With static mounts, clients will have to be individually unmounted from the old storage. As with **Forward Relocation**, the old server's showmount can be used as a starting point for finding clients who need to be unmounted.

Case Studies

Case 1: Northrop Grumman Corporation

Seminal Case: The Genesis of On-line Relocation

The Northrop Grumman case was the genesis of **Full On-line Relocation** techniques. On the system there, the multi tiered HSM software had deadlocked at the first HSM level. This level filled to capacity and was unable to migrate files to deeper levels and, lacking space, was unable to retrieve files from the next deeper tier. With the backing store locked up, continuing file creation on the primary media caused them to fill to capacity and refuse to take new data. With the primary media full and the first tier storage deadlocked, no retrievals of staged-out files could be completed. Any client systems that referenced staged-out files or tried to create new files would suffer permanent NFS timeouts (they used hard mounts for robustness) and would eventually hang.

A replacement fileserver large enough to accommodate all of the data was already in place, but there seemed to be no way to relocate the data to it. At that time, over a hundred thousand of the half million files managed by HSM were in this state. On a multi-vendor UNIX LAN of around 200 regular users, at least twenty percent (20%) of the workstations had to be restarted each day to circumvent HSM NFS hangs. Something had to yield.

After a month of trying to repair the system, Freyburger and Cordrey were seeking a way to avoid abandoning all of the inaccessible data. *That* was when the innovation to relocate the files on-line by replacing them with symbolic links was made. At the time, all issues of losing small amounts of data because of race conditions became secondary to recovering as much inaccessible data as possible and resolving client system hangs. All resident files were relocated one at a time to the new server and replaced with symbolic links in the hopes that some of the inaccessible data could be retrieved once space became available on the old storage.

In the first phase, find and the vendor supplied version of ls were used to identify resident files, with cpio being used to relocate them. As it turned out, freeing space on the primary media was enough to relieve the pressure on the HSM system. In the process, more and more files that had previously been inaccessible became available again. It was also necessary to manually recreate the HSM databases several times as the filesystems were evacuated, but that was a well documented process, already in the manuals supplied by the HSM vendor. This phase alone was sufficient to completely evacuate one of the filesystems.

In the second phase, a script was written to iterate through the database for each filesystem and force relocate those staged-out files that were local to the first tier HSM storage. Since only a few dozen files remained when this phase was completed, ''fingerprinting'' techniques were used to locate, for recovery by hand, those last few files.

No third phase was needed to recover files from the second tier HSM storage because the evacuation was complete. This was despite the fact that the robot was 80% full – all of the data in the robot was stale, representing deleted and prior versions of current files, because garbage collection had never been done.

Race Conditions and Problems

No data was lost to race conditions! Some users even ran make and similar programs in their directories while those directories were being swept clear of files. Since the relocation involved about a half million files in active use by two hundred developers, this came as a pleasant surprise to Freyburger and Cordrey.

One unsurprising anomaly was encountered: some executables (web server daemons in this case) exited when their binaries were relocated.

During the development of the software to do **Full On-line Forward Relocation**, two main problems were encountered: **Double Relocation** and **Pathological Filenames**, both of which are discussed in the previous section.

Non-HSM Filesystem Relocation

There were two filesystems on the old servers that were not HSM managed or had never had files staged-out. Since the servers were slated for decommissioning, that data had to be relocated as well. One of those two contained several web sites which supported the entire corporation, so it had to be available at all times with no outage.

Having used **Full On-line Forward Relocation** on filesystems under HSM management, the authors applied their software to those normal and critical filesystems. The relocations completed in a single pass, during the production day. Further, because the binaries for the http daemons were stored local to the front end web server which served its content from the NFS mounted volume, the daemons continued serving with no interruptions due to having their binaries moved out from under them.

Case 2: Hughes Space and Communications

As part of a standard technology refresh program, an obsolete, non Y2K compliant Convex 3240 was replaced and the filesystems on it were rehosted to a new file server. Most of the data was destined for a read-only section of the new storage, while home directories were placed in a filesystem with strictly enforced quotas.

Non HSM File Systems

One filesystem had never staged-out files to HSM tapes. This filesystem was evacuated to the read-only ''legacy'' disk space in one pass using **Forward Relocation** during the production day.

Home directories were handled differently. Since large numbers of project files had been stored under home directories, most of those files were destined for read-only storage. Only dot files and dot directories were relocated to the new, read-write, home directory storage. Similar to a **Reverse Relocation**, directories and files appearing in the top level of each user's home directory were pre-populated with reverse links pointing to the old storage. The old storage was set to read-only mode by management request. After this, the users were free to replace those links with actual files; the reverse links had been created to reduce the impact of home directory relocation.

HSM File Systems

The system was still working, but the media and tape drives were aging and failing. It was also too slow for users to relocate their own data. Due to the extremely large number of files (approximately 997,000) pre-populating the new storage with that number of symbolic links was not practical. Therefore, to minimize user impact, relocation of resident files was started using **Forward Relocation**.

Shortly after the data movement began, management requested that the old system be placed into a read-only state. This changed the availability state to **Semi On-line**, and data relocation continued using the **Forward Relocation** algorithm.

Relocation was paused after the resident files were completed. The Convex was taken **Off-line** and the primary reference point for the user community became the new ''legacy'' mount point. Shortly thereafter, the HSM system was rehosted on a physically smaller system.

Following rehosting, the new *old* server was placed **On-line** in **Read-Only** mode. In this optional step, a **Hybrid Linkder** populated the ''legacy'' storage with **Reverse Relocation** links. This permitted users to read copies of their files even before they were relocated, reducing the burden of by hand recovery and relocation.

Data movement resumed using **Reverse Relocation** by retrieving all files on a particular medium and feeding their names to the relocation worker program. Some repair of damaged tapes was done and files from the repaired tapes were retrieved. Backup tapes were also used to retrieve files whose HSM tapes had degraded beyond usability.

On the new storage (a read-only legacy filesystem), files older than about one year were archived to DLT tape. Once archived, these files on the new storage were replaced with symbolic links pointing to the nonexistent object, `archive`, so that users browsing the filesystem would be able to view the names of all files available.

Case 3: RAND

Non HSM Home Directories: Forward Relocation by Request

All UNIX account home directories (over a thousand) resided on non-Y2K compliant servers, which had to be upgraded as a part of a standard technology refresh program. The replacement servers were separate fileservers with large RAID boxes at each campus.

`mh` is used pervasively at RAND, thus, because the resynchronization of the source and destination file systems was not built into Version 2 of the **Forward Relocation** algorithm, it was not used to move UNIX account home directories to the new servers.

However, users could request an early relocation of their home directories by contacting their help desk. As part of this standard help desk procedure, **Full On-line Forward Relocation** *is* used to move their home directory to one of the new servers.

*HSM systems: **Enhanced** Just Plain Copy*

This HSM exit was accomplished by Weissler. It is included for completeness to demonstrate that highly successful **File System Relocations** do not require **Forward** or **Reverse Relocation**.

RAND acquired two Epoch optical hierarchical storage management systems in 1989-1990. The initial systems were Sun 4/75 workstations with a proprietary Epoch operating system based upon SunOS 4.0.3. Epoch used Ingres as the supporting relational database with Hewlett-Packard and Hitachi optical Jukeboxes populated with WORM [write once, read many] media. A series of upgrades brought the systems up to Sun Sparc 20 workstations running SunOS 4.1.3 with erasable optical media.

By 1996, it was clear that the systems would have to be replaced. Backups had become increasingly difficult as the amount of data increased: it was common for a full backup to run several days, rendering it of questionable integrity. Staff turnover left RAND with little expertise in HSM which in turn led to the deterioration of administration. On going garbage collection efforts decreased with the staff turnover, resulting in many of the 1200+ optical media being under 50% utilized. The vendor stopped supporting the non-Y2K compliant hardware which rendered relocation of the data mandatory.

The system was running, healthy, old, and slow. Because it was healthy, a PR campaign was necessary.

Some users were convinced to buy their own disks, some wanted the "higher performance" of not having to wait for stage-ins, and others had to be shown the lower overall support cost of newer technology storage systems.

Analysis was conducted to find usage patterns. Three patterns emerged and data that followed these patterns was copied to different target servers. However, because the replacement servers did not arrive on site at the same time, a systematic draining of each optical media was not possible since only a portion of each media could be relocated to a given server. The first server arrived and a portion of the data was moved, and the process was paused. Relocation temporarily resumed after the second server was delivered. When third server arrived, the last of the data was evacuated, some three months after the process began.

The system was placed in a **Read-Only** or **Semi On-line** state during the relocation. The Epoch utility, epls, worked rapidly enough to allow media preparation in the form of pre-load lists per directory. To avoid thrashing, *all files* were staged-out to optical storage leaving the file systems largely empty. Files in the pre-load lists were then staged-in in bulk using epbsi. Data was then manually copied using tar in relatively small chunks. These improvements on the **Just Plain Copy** technique virtually eliminated jukebox thrashing. This makes this example much higher performance than a brute force **Just Plain Copy** approach.

Summary

The **Forward** and **Reverse** algorithms described in this paper offer a different approach to data relocation that does not appear to be in common use. Since they offer options for providing data availability during the relocation process, there are benefits to be reaped by sites choosing to employ these methods. The intent of the authors is to seed these techniques into the thinking and planning of Systems Administrators and Managers.

Performance Comparisons

The performance of an **On-Line** fileserver is *infinitely* higher than the performance of an **Off-Line** fileserver. While the **On-Line Relocation** methods presented here take longer to run on a fileserver when compared with previously available methods, those previously available methods generally require filesystems to be made unavailable to users during the copy. At sites with high uptime requirements, no comparisons of wall-clock times are relevant.

Since **Reverse Relocation** requires **Read-Only** mode or some downtime, it suffers the same problem as previously available methods, and should only be used at sites that can tolerate these changes in data availability (generally lightly used HSM systems or non-healthy servers).

Non-HSM Uses

Server replacements can be done during the production day using a **Full On-Line** technique. As the availability of large capacity disk systems brings them into wide deployment, the time required to relocate the files from one server to another is becoming longer. With these long duplication times, and availability demands, **Full On-Line Relocation** algorithms can be used to reduce the impact of such transitions by minimizing outage windows and allowing data to be relocated during the production day. These techniques can even be used to relocate files within the same server as would be required to move from a traditional filesystem to a journaled filesystem or to relocate large directory trees. Given the choice of large outage windows or near 100% availability, some administrators can reduce the impact of their relocation projects with techniques similar to these **Forward** and **Reverse** algorithms.

Limitations

These techniques are not appropriate for all filesystem relocations. They fail to prevent stale NFS handle errors for environments where the directories are held open by a process being cd'ed there for long periods of time. Files used by programs that keep them open for a long time and change them regularly, like databases, are likewise inappropriate.

The construction of the programs required to perform these tasks is within the capabilities of Sage Senior (Level 4) administrators. However, caution and forethought must be applied to the construction of the code to avoid the pitfalls of double relocation and pathological filenames.

Ongoing Work

Work is under way to enhance the algorithms by recoding them in C++ with advisory file locking and post copy check summing. This work will also be published with significantly expanded pseudo code and will include a full discussion of when a symbolic link may be safely relocated.

Author Information

Vincent Cordrey <cordrey@acm.org> first experienced UNIX in 1981 on a PDP 11/45 running Version 7. He did Systems Administration and wrote custom business software from 1984 through 1987, porting the solution to UNIX in 1988. That was when his work became almost exclusively UNIX Systems Administration.

Doug Freyburger <freyburger@ieee.org> started in the computer industry in 1978, working on projects from custom VLSI design for spacecraft at the Jet Propulsion Laboratory to stereoscopic video games for a start-up. In 1986, after doing Systems Administration as a sideline for five years, he switched to doing it full time, and has been at it ever since.

Jordan Schwartz <jordan@colltech.com> started his career in data processing as a third shift computer operator at RAND in 1989, and was promoted to the Systems Administration group in 1993. He has been a consultant with Collective Technologies since 1998.

Liza Weissler <liza@colltech.com> worked as a technical writer at Systems Development Corporation and RAND, but moved to Systems Administration at RAND in 1987 when she decided it was more interesting to do things rather than write about them. She joined Collective Technologies in 1999.