# Hiding HSM Systems from the User

Hartmut Reuter
Rechenzentrum Garching der Max−Planck−Gesellschaft (RZG), Germany
hartmut.reuter@rzg.mpg.de

## Abstract

File life−times longer than the life−time of HSM−systems make it necessary to move files from one HSM−system to another. Such a migration takes a long time and cannot be hidden from the user unless an additional layer of software is put in between. RZG is using MR−AFS as such a layer. The features of MR−AFS are presented once again. RZG's experiences with HSM−migrations and the usage patterns of MR−AFS are discussed.

## Introduction

Today not only high energy physics or science in general, but also the digitalization of libraries, archives and any kind of documents produce a vast amount of data which has to be kept for a very long time. Much attention is being paid to the question of how to organize the data in such a way the information can be retrieved later.

Here another − more trivial − problem is addressed: file life times in these archives exceed life time cycles of HSM systems. Thus the files have to migrate during their life time several times to new HSM hard− and software. While on most HSM−systems a migration to new peripherals such as new tape drives and media can be handled transparently to the user, it is not that easy to hide the migration to a new HSM system.

Replacing a mass storage system by a new one is a long running process because the data have to be brought on−line and copied. Such a migration may take several months to complete. While the migration is in progress part of the files are still in the old system and part already in the new one. The user having access to the data in the HSM system by the "classical" means of FTP or NFS will be affected by the migration because the path−names of his files change. The situation is even worse for application software where a concept of data access based on path−names was implemented.

Therefore application software and users must be screened from the knowledge about details of the HSM environment. This can only happen if an additional software layer exists between the HSM system and the user or the application. This layer could be a database system in which the actual location of each file is stored and updated during the migration process. Using a database is a typical approach for data mining software, but the more general solution is to have a filesystem as the intermediate layer.

AFS and DFS are both filesystems with an architecture that in principal would hide details of the underlying HSM systems. Presently, however, AFS in its special implementation called MR−AFS is the only one to really do that. The HSM interface used by DFS [1] is still too simple, but additions to its concept can be thought of which would allow for the same functionality MR−AFS already has. The following therefore will concentrate on MR−AFS.

## Multiple Resident AFS

MR−AFS (Multiple Resident AFS) is a distributed hierarchical filesystem with the client side being Transarc's widely spread AFS while the server side was developed at Pittsburgh Supercomputer Center (PSC). The server side also started from Transarc's source code, but the number and amount of changes and additions is such that the original code now is less than half. MR−AFS was presented in talks at the 1993 and 1994 IEEE mass storage symposia [2] [3]. In 1995, however, the three main authors (Jonathan Goldick, Bill Zumach and Chris Kirby) left PSC, two of them to join Transarc. Since then MR−AFS has been maintained and further developed by RZG.

### The concept of shared residencies

Shared residencies which gave the name to MR−AFS are disk partitions which can be shared by all MR−AFS fileservers. If e.g. during the writing of a file a fileserver sees a shortage in disk space in the partition where the AFS−volume lives, it can write the file instead into one of the shared residencies which may be local or on another machine. In order to allow fileservers to do I/O operations on another machine a lightweight daemon called "remioserver" must run on that machine. The fileserver communicates with this remioserver by remote procedure calls of the same kind as those used between the AFS−client and the server. A shared residency can also be a HSM partition on an archive−server. This is called an archival residency in the following.

The way HSM−systems work could, however, conflict with the technique used normally by AFS to store files and directories. This is because the files and directories belonging to AFS−volumes are not visible in the disk partition. Visible is only one small file per AFS−volume which contains inode numbers leading to the metadata files. From the metadata files the AFS fileserver extracts the inode numbers of files and direc−

tories inside the volume. To open a file or a directory the fileserver uses a special AFS−system−call based on the inode number rather then a path−name in the file−system's directory structure. Additional conflicts could arise from the fact that the AFS−system−call abuses some fields in the inodes of the filesystem to store additional (redundant) information.

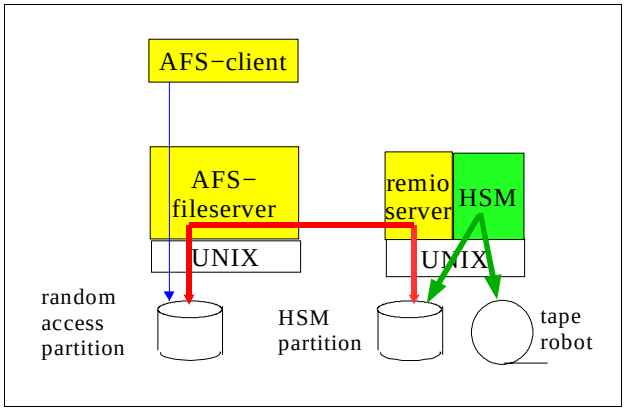


*Figure 1: The AFS−client is served from a random access residency (blue arrow). The fileserver copies files to the HSM−partition (red arrow) where the HSM−system writes them onto tape (green arrow).*

For HSM−partitions, therefore, MR−AFS uses simply the Unix filesystem as provided by the vendor. In such a partition a directory tree of up to 256 * 256 directories is built and the AFS−files (or directories) are distributed in the tree by a hash algorithm and stored with names built from their bitfile−ids. By this technique the HSM−system needs to know nothing about AFS and vice versa. The only thing AFS must know about such a partition is that opening of a file may take a while. Table 1 shows HSM systems that have been used with MR−AFS.

| ***HSM−system*** | ***OS*** | ***AFS−cell*** |
|---|---|---|
| ADSM | AIXe | cpc.engin.umich.edu |
| DMF | UNICOS | ipp−garching.mpg.de<br>psc.edu |
| DMF | IRIX | ipp−garching.mpg.de |
| EMASS Fileserv | IRIX | federation.atd.net |
| Epoch | ? | ? |
| SamFS | Solaris | tu−chemnitz.de |
| Unitree | Solaris | rrz.uni−koeln.de<br>urz.uni−magdeburg.de |

*Table 1: HSM−systems used with MR−AFS, the Epoch−support is implemented in the source code, but it is not known where it has been used.*

For some of these HSM−systems 64 bits of the HSM−metadata are copied into the AFS−vnode describing the file copy in the HSM−partition. This was primarily done to add some redundancy, only later it turned out to be very helpful in the process of migrating from one HSM system to another.

**The residency database**

Information about shared residencies is provided by an additional database on the AFS database servers. For each shared residency, the following must be specified

- accepted file size range,
- priority,
- whether it's wipable (see below),
- wiping thresholds,
- wiping weight factors,
- minimum migration age,
- and others.

The "minimum migration age" is used for residencies on archival servers. On residencies for which the "minimum migration age" is set the fileservers automatically create copies of all files which have reached this age. These copies on archival residencies can be used as backup copies of the files, but they also allow for a MR−AFS internal data migration.

**Wiping: the internal data migration in MR−AFS**

The copies of files on an archival residency (typically on tape) allow MR−AFS to do its internal data migration: local partitions and shared residencies on random access storage can be declared wipable. Files in wipable residencies may be wiped away if they exist in another residency elsewhere. Wiping is controlled by high and low water marks specified in the database. If the high water mark is reached the fileserver removes files which have another residency until the disk usage is under the low water mark. The algorithm by which the fileserver selects candidates to be wiped from the partition is also defined in the database and can be set individually for each shared residency.

If an AFS client requests data from a file with only an archival residency, the fileserver copies the file to random access storage before delivering any data. The new random access residency preferably is chosen on the same fileserver where the AFS−volume belongs in order to avoid unnecessary network traffic.

**The fetch queue**

The open for read of a file on an archival residency can take a long time if the file has to be brought on−line by the HSM−system. To avoid being blocked in the

open system−call during this time, MR−AFS forks a new process to do a primary open. Only after this process has succeeded, the remioserver will try to open the file. The remioserver keeps information about these children in the fetch queue which is also used to schedule fetch requests in order to optimize throughput on a fair basis.

The response time of the HSM−system to bring files on−line is strongly non−linear with respect to the number of requests being processed. MR−AFS therefore provides a queuing mechanism in order to handle situa−tions of high load in a graceful way. The position in the fetch queue where the new request goes depends on the

```
~/texte: fs fetchqueue
Fetch Queue for residency backup (4) is empty.
Fetch Queue for residency stk_tape (16):
Pos. Requestor  FileId                      TimeStamp    Rank Cmd. State
   1 ata        536880083.9098.75195     Nov  3 16:36   0 open waiting for tape
   2 hwr        536879945.36.30569       Nov  3 16:37   0 pref waiting for tape
   3 hwr        536879945.400.48423      Nov  3 16:37   1 pref waiting for tape
   4 hwr        536879945.270.55867      Nov  3 16:37   2 pref waiting for tape
   5 hwr        536879945.382.46363      Nov  3 16:37   3 pref waiting for tape
Fetch Queue for residency d3_tape (64):
Pos. Requestor  FileId                      TimeStamp    Rank Cmd. State
   1 rln        536878253.30966.113655   Nov  3 16:33   0 pref waiting for tape
   2 afsbackup  536911346.107946.80670   Nov  3 16:34   0 open waiting for tape
   3 hwr        536879945.114.55860      Nov  3 16:37   0 pref waiting for tape
   4 afsbackup  536911349.52448.39391    Nov  3 16:34   1 open waiting for tape
   5 hwr        536879945.474.48661      Nov  3 16:37   1 pref waiting for tape
   6 hwr        536879945.234.53602      Nov  3 16:37   2 pref waiting for tape
   7 hwr        536879945.358.55885      Nov  3 16:37   3 pref waiting for tape
   8 hwr        536879945.370.56301      Nov  3 16:37   4 pref waiting for tape
~/texte:
```

*Figure 2: Output of the "fs fetchqueue" command. On 2 of the 3 archival residencies are fetch requests queued. All requests are waiting for the HSM−system to bring the file on−line.*

number of requests the user has already in the queue, thus allowing for a fair share scheduling of HSM re−quests. This technique has been adopted from RZG's old main−frame based HSM−system HADES [4]. The queueing mechanism also controls the number of HSM requests being processed in parallel. This avoids a con−gestion of the HSM system and allows, on the other hand, enough parallelism to read multiple files from a tape once it has been mounted.

This parallelism increases the throughput of the HSM−system significantly. Therefore the user is en−couraged to start the prefetching of all the files he needs in a session or a batch job at the beginning. The management of the fetch queue guarantees that a user with a high number of requests cannot block the requests of all other users. MR−AFS allows the user also to inspect the actual fetch queue by a command he can enter on his AFS−client machine (Figure 2).

### Scalabilty by use of multiple archival residencies

Small sites and MR−AFS beginners have a single archival residency where all files bigger than a certain size are archived. But with the growth of data in many cases it makes sense to separate different file−size ranges to individual archival servers with different tape hardware.

Another reason for having multiple archival resi−dencies can be scalability. New experiments in high en−ergy and plasma physics are supposed to produce data at a rate of one to several TB per day. Here MR−AFS could also be used as a common filesystem even if the data streams have to be directed to separate archive servers. The scalability of AFS in general is guaranteed by the fact that the fileservers are completely independent from one another and that the metadata needed to build the AFS−tree on the client machine are almost completely decentralized in the fileservers. The same is true for MR−AFS if you ensure that each fileserver has a preference for a dedicated archive server. This can already be enforced by masks for desired and undesired residencies which are part of the AFS volume metadata. To separate the different data streams in order to avoid congestion is then only a question of network topology.

Scalability is a very important issue today and with MR−AFS it can be achieved with rather simple and cheap means without using a single huge high−end HSM system such as HPSS.

## Configuring a MR−AFS−cell

Since file copies on archival residencies generally are moved off to tape by the underlying HSM system, these file copies should be considered as tape−files. MR−AFS then is more a convenient way to create and access files on tape rather than as a kind of infinite disk space. That means MR−AFS should be used only by sites which have real mass storage requirements. Even for those sites, wiping should be limited to the kind of files you would store on tape, anyway.

All other files should not be wipable because it is faster and in many cases also cheaper to have them on disk. This goal can be reached in different ways:

- The simplest way would be to configure only one shared residency which is the archival residency. If the file−size range for this residency starts at a rea−sonable size for tape files, say several MB, wiping can be allowed on the local (non−shared) disk parti−tions of the fileservers. The disadvantage of this ap−proach would be that it would take the fileserver a long time to get the list of candidates for wiping be−cause also the small files (which are safe from being wiped because they don't have additional residencies) are in the same partition.
- You then could separate the small files from the big ones by having a special shared random−access resi−dency for big files on each fileserver and declaring only these wipable. If the size−ranges allowed for

the local disk partitions and the shared residencies are the same on all fileservers, volumes may be moved around between all servers. Then all files bigger than a certain threshold are wipable and below not, no matter on which fileserver and where in the AFS−tree they are.

- But it could also be desirable to allow wiping only in certain subtrees of AFS and to concentrate wipable files on a few fileservers. This is the configuration RZG has chosen. The advantage is that the user can trust that the files in his home−directory and all the software is on−line and only in well−known parts of the tree files may be on tape. The disadvantage then is that you cannot move each volume to each fileserver, but you have two classes of fileservers.

In our case only big files originating from observations in plasma physics and X−ray astronomy or from supercomputing batch jobs are wipable. These files are stored on four different fileservers each with 40 to 100 GB of disk space, but the total size of the files stored per fileserver varies beween 1.5 and 2 TB. The AFS−volumes on these fileservers are mounted in special subtrees in AFS for which everyone knows that access to files may be slow.

One additional wiping fileserver provides the general user with HSM storage for archiving: RZG gives each user additional to his non−wipable home−directory in AFS a private volume mounted in a special "m−tree" in AFS, "m" for migrating.

Our AFS−cell "ipp−garching.mpg.de" runs 20 fileservers and five database servers. To simplify maintenance all fileservers run the same MR−AFS−binaries, but only six of them make use of the special features of MR−AFS, all others behave like standard AFS fileservers.

The reason for the high number of database− and fileservers is that the cell extends over some remote sites 500 to 800 km away from Garching. To make sure that in the case of a network outage these remote servers can still serve the local AFS−clients a read−only copy of the database server runs at each remote site.

## Experiences with HSM−migrations

RZG has been running MR−AFS since 1994. In the beginning the HSM system being used was DMF on a Cray−EL with robot attached D2−drives. In the following years two migrations of files from this HSM system to new ones took place:

- 1996 all tape copies of files < 8 MB were moved from D2−tapes connected to the Cray−EL system to STK Timberline tapes connected to a Cray J90 system (~ 350,000 files, 0.5 TB, 6 months).
- 1998 the remaining files on D2−tapes were moved to STK Redwood−tapes connected to a SGI Origin 2000 system (~ 120,000 files, 3.2 TB, 4 months).

After creation of a new archival residency all on−line files and files being brought on−line which match the characteristics of this residency automatically get copies there. But then still a large number of unfrequently or never used files remain on the old archival residency. One could think of bringing them on−line by prefetching them, but then these inactive files would fill up the fileserver partitions, wiping off the files the users are really using. Therefore and for performance reasons something more needs to be done.

During both migrations a table of contents for each tape was extracted from the DMF−database. This table contained for each file on the tape only a so called "file−handle" instead of the real file name. The "filehandle" is also stored in the inode of the file. In the DMF case the HSM metadata mentioned before that is stored in the MR−AFS volume metadata is exactly this "filehandle". This allowed us to identify the files in MR−AFS. On the Cray system a program was then run for each tape which brought the files on−line in the order they were written on tape and which triggered MR−AFS to add a new residency on the new HSM system for that file and delete the one on the old HSM system. 50 of these DMF "dmget"−requests and three of the add− and delete−residency commands were run in parallel in order to minimize the number of tape mounts and maximize the through−put. The add−residency command moves the data directly from the old to the new archival residency without affecting the random access residencies used as staging space on the fileservers. During this whole procedure the AFS users had no interruption in service.

In spring 1998 the STK−Timberline tapes were moved from the Cray J90 system to a second SGI Origin 2000 system and the whole DMF−system was converted from Cray−DMF to SGI−DMF format. In this case the service was interrupted for one day and the only change in MR−AFS was to update the residency database entry

| | ***non−wiping (standard AFS like)*** | ***wiping MR−AFS*** |
|---|---|---|
| ***Fileservers*** | 14 | 6 |
| ***Files*** | 6.2 million | 1.6 million |
| ***disk space*** | 0.5 TB | 0.5 TB |
| ***total data*** | 0.4 TB | 7.6 TB |
| ***average file size*** | 64 KB | 4.9 MB |

*Table 2: Non−wiping and wiping fileservers. The approximately 600,000 files in read−only AFS−volumes are not shown. That accounts on the big difference between disk space and total data for the non−wiping fileservers.*

for this residency.

## Usage patterns of MR−AFS

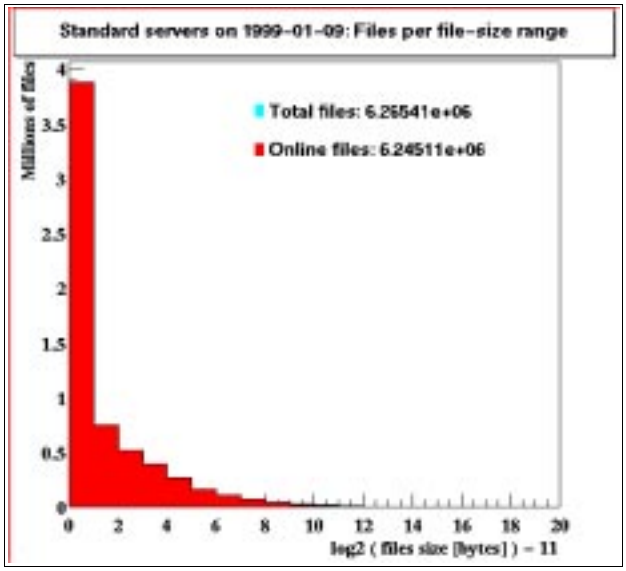


*Figure 3: Distribution of files per file−seize range on non−wiping AFS−servers (columns for [0−4KB] up to [1GB−2GB]).*

Figure 3 shows that more than 50 % of the files on non−wiping AFS−fileservers are smaller than 4KB. This is what can be expected for Unix home−directories. For this kind of file−size distribution data migration doesn't make sense.

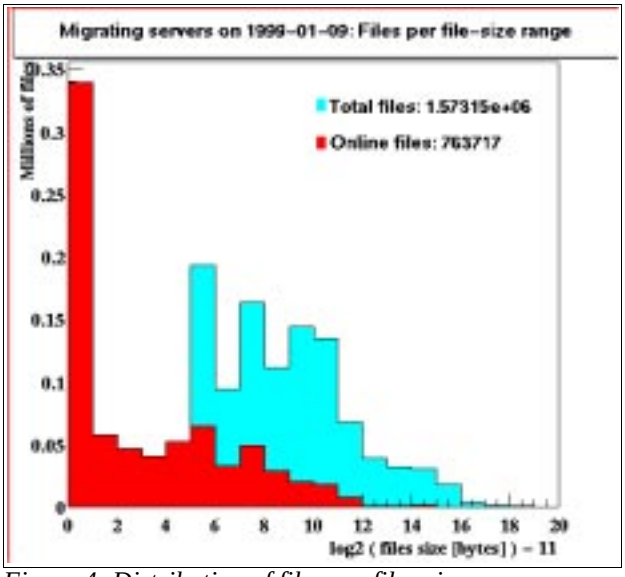


*Figure 4: Distribution of files per file−size range on wiping MR−AFS−servers.*

On the wiping MR−AFS−fileservers (figure 4) the file−size distribution maximum is shifted to much higher files−sizes. There are, however, still a lot of small objects. This has two reasons:

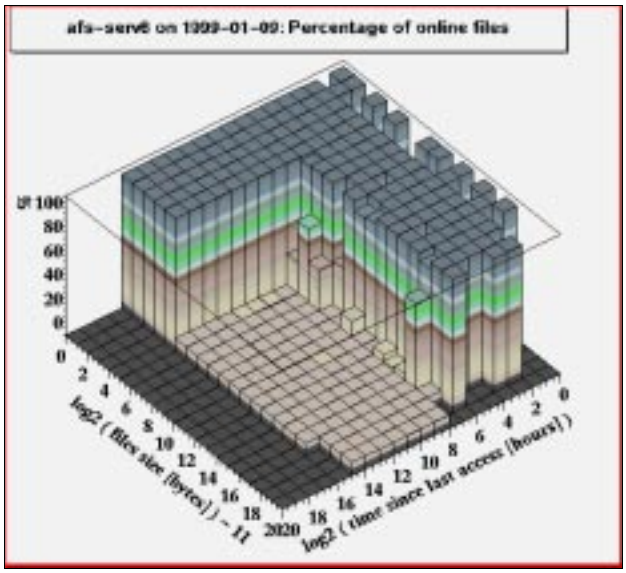


*Figure 5: Percentage of on−line files per file−size range and per time since last access. Missing values are negative (dark).*

- AFS makes no difference between files, directories, and symbolic links, so all these objects appear as files.
- In the beginning users created in their "m−tree" lots of small files before we introduced a quota for the number of files in these volumes.

The residency database allows for each shared residency to specify a lot of weight factors to calculate the order in which files should be wiped. Our current policy does not make use of all these possibilities, but simply takes the time passed since the last access of a file as the main criterium. Therefore we see in figure 5 for bigger files a sharp life−time on the staging residency − in this case of 2**6 = 64 hours. The files smaller than 2**(5 +11) = 64 KB are always on−line. The dark stripe in the foreground is a region for files older than the 2**15 hours which exceeds the age of this server.

Figure 6 shows that 2/3 of the files in the "m−tree" have never been read. For the other wiping servers which are dedicated to individual projects this fraction is much lower, however.

Figure 7 shows the total data of all files in our cell by file−size range. The maximum is in the file−size range of 64 − 128 MB. The on−line portion of all data (red) is obviously rather small (< 10%) while the online portion of files is about 85 %.

In the statistic over the wiping fileservers the percentage of on−line files (figure 8) shows that files

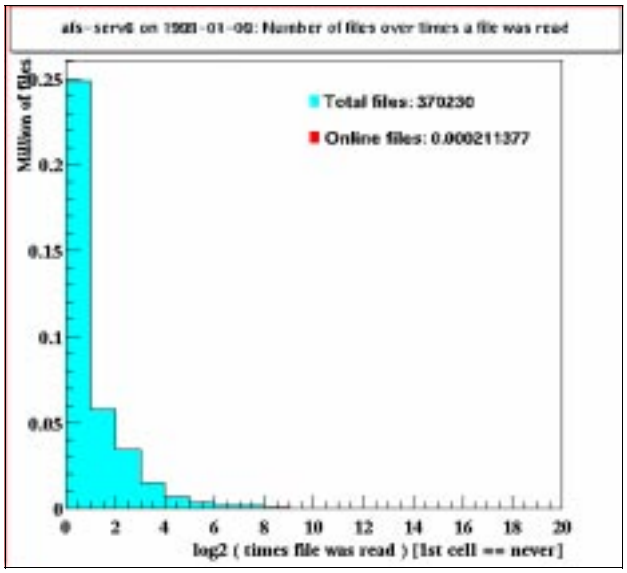


*Figure 4: Files on the fileserver for the "m−tree" by number of times they have been read. First column means: never.*

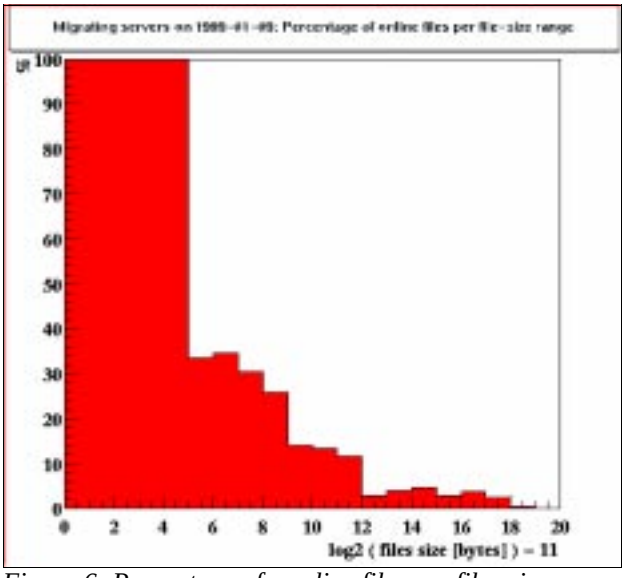


*Figure 6: Percentage of on−line files per file−size range on wiping fileservers. All files < 64 KB are on−line because they are stored in the local disk partition.*

smaller 64 KB could always be kept on direct access storage. The bigger the files the less probable it is to find them on disk. This is remarkable because our migration policy is based primarily on the last access time rather than on file−size. Therefore it must be the user's access pattern which prefers smaller files.

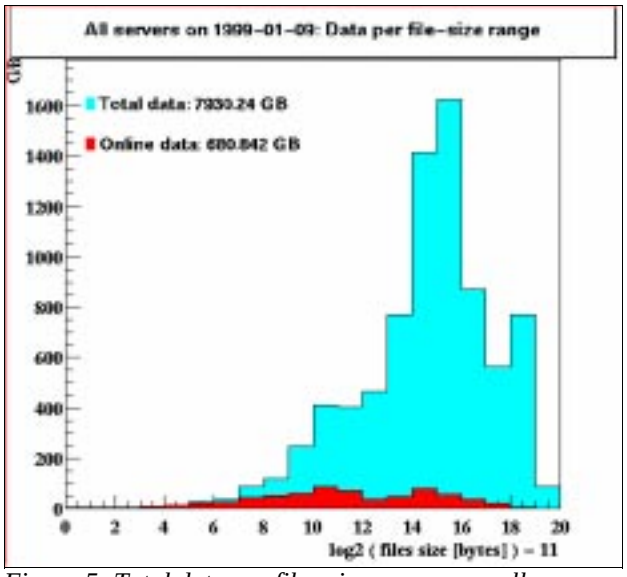


*Figure 5: Total data per file−size range over all fileservers in the cell. The blue columns represent the total data, the read ones the on−line portion.*

## Retrieval and staging times and reliability

We see a staging from tape to disk of about 10 GB/day with peaks up to 50 GB/day. Transfers from disk to tape are more than twice that high.

| ***Tape drive*** | | ***Daily average of HSM retrieval time in last 61 days*** | | |
|---|---|---|---|---|
| ***Type*** | ***number*** | ***minimum*** | ***median*** | ***maximum*** |
| Timberline | 7 | 23 sec | 45 sec | 142 sec |
| Redwood | 8 | 107 sec | 297 sec | 8343 sec |

*Table 3: Daily average of the retrieval time of files in the HSM system. Even in the best case longitudinal recording technique is 5 times faster than helical−scan technique.*

As indicated before retrieval times are strongly nonlinear to load. They also depend on the kind of tape drives used. In terms of reliability we see a significant difference between longitudinal and helical−scan recording technique. Helical−scan tape drives turned out to be very sensitive to the quality of the media. We have never lost any data (having configured the HSM−systems to keep two independent tape copies of each file), but the retrieval times sometimes were unacceptably long due to broken tape drives or positioning problems. Also the high capacity of helical−scan media sometimes lead to long retrieval times if a file is requested from a tape which at the same time is being written. This, of course, is weak design of the HSM software.

Staging life−times depend nonlinearly on the relation between the data volume needed at a time and the disk space available. We see random access residencies with file life−times of months and those where the average file life time is shorter than two days.

These very short staging life−times are not a problem for the user community of this fileserver because the files are read only once during a batch job. Each batch job needs other files and prefetches them. So it is just the classical tape processing such as in the old mainframe world.

The average time between software crashes is in the order of some months. Therefore nearly all service interruptions are caused by hardware failures such as of tape drives, network interfaces, disks, and routers (power failures do no harm because all servers and tape robot systems are battery buffered). Of course, users with only on−line data are much less affected by failures than those who need off−line files.

## Conclusion

MR−AFS has proven to be mature, stable. It is the common filesystem for Unix, Windows NT (and via Samba also for Windows95) at RZG. The reliability of the whole system is limited mainly by the reliability of the underlying hardware. The MR−AFS internal data migration (wiping) gives better control over the behaviour of the system than can be achieved with most vendor's HSM solutions.

For the following reasons MR−AFS can be used as a common scalable filesystem up to far higher capacities then are being used today:

- AFS is inherently scalable due to the independency of fileservers from one another and the decentralization of the metadata (no hot spots).
- New fileservers for on−line disk space and new archival servers connected to HSM systems for the archiving can be integrated at any time.

Finally it allows one to use different vendor's HSM−systems at the same time and to exchange them without interference with the user's view of the filesystem. This is very important because it allows one in the future to revise today's decisions for HSM solutions.

## References

[1] C. Everhart, *HSM for DCE DFS*, Proc. Decorum '96, February 1996.

[2] J. Goldick, K. Brenninger et al., *An AFS−Based Supercomputing Environment*, Proc. Twelth IEEE Symposium on Mass Storage Systems, April, 1993.

[3] C. Maher, J. Goldick et al., *The Integration of Distributed File Systems and Mass Storage Systems*, Proc. Thirteens IEEE Symposium on Mass Storage Sys−tems, May, 1995.

[4] H. Reuter, *The HADES File Server*, Proc. Eleventh IEEE Symposium on Mass Storage Systems, October, 1991.