Can you run ADSM in an AFS environment?

(ADSM in the URZ Heidelberg)

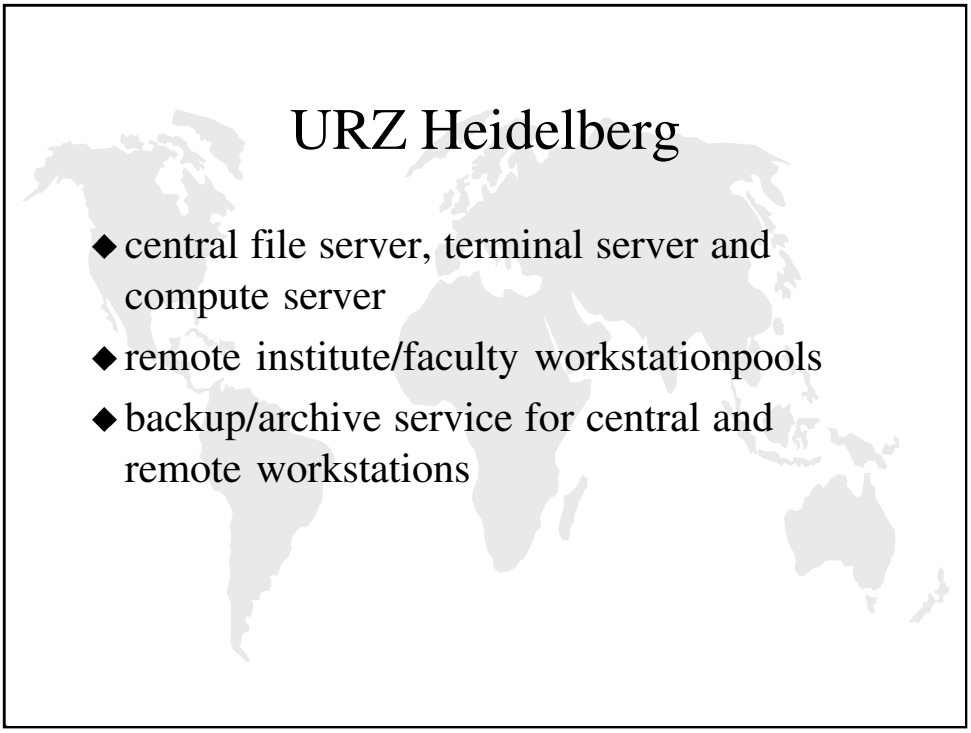
Rolf Bogus, URZ Heidelberg

2nd ADSM Workshop, April 1/2, 1996

Overview

- URZ Heidelberg and ADSM
- Short introduction to AFS
- Problem 1: Large Filesystems
- Problem 2: Large number of filesystems
- Problem 3: Many workstations use the same filesystems
- Problem 4: Backing up and restoring AFS ACLs
- Conclusion

After a short description of the Universitätsrechenzentrum Heidelberg and an introduction of AFS (the Andrew file system) the main problems of using ADSM in a distributed (AFS) environment will be presented.

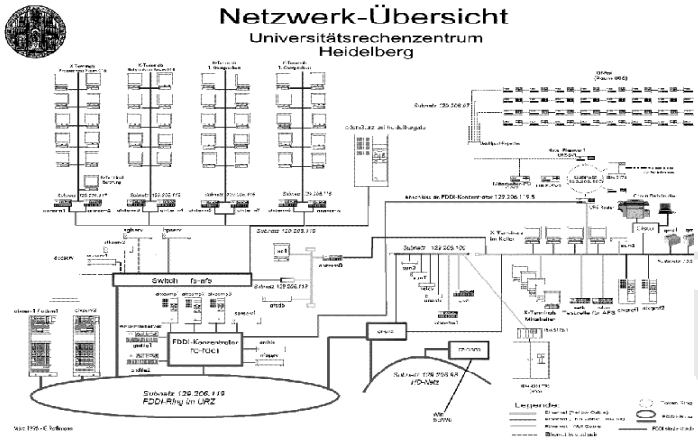


The Universitätsrechenzentrum (URZ) Heidelberg operates a UNIX workstation cluster (mainly IBM RS/6000, but also workstations of all other producers) with two AFS file servers containing about 60 GByte data, 9 terminal servers, three IBM and some other compute servers. This UNIX workstation pool is the subject of this talk.

Other faculties and institutes run their own workstation or PC pools.

The URZ offers a backup/archive service for the remotely operating workstation and PC clusters using ADSM.Those mainly small pools use ADSM as ADSM was designed. They don’t make problems.

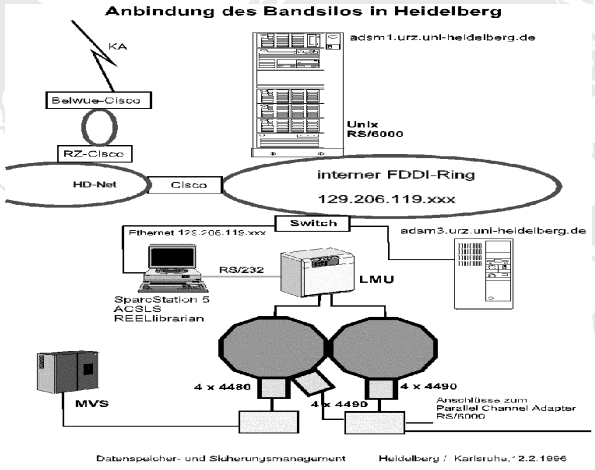
URZ Heidelberg: Net



UNIX users working in the building of the URZ can choose between three terminal rooms with 9 terminal servers. The user data reside on 2 AFS file servers with two RAID arrays containing about 60 GByte disk space.

These two file servers are backed up by ADSM.

URZ Heidelberg: ADSM



The URZ Heidelberg operates now one production ADSM server (still version 1.2.0.9 because of a database problem, the ADSM development team is working on) and one ADSM server for the disaster recovery project with the university of Karlsruhe.

Tapepool data is stored in two StorageTek ACS 4400 silos with 6 dedicated 3490E drives driven by two parallel channel adapters. Mounts are managed by a SUN SparcStation 5 running the ACSLS software.

Plans for 1996 include two more ADSM severs with StorageTek Redwood and Timberline drives.

Short introduction to AFS

- Global (world wide) filesystem.
- Not the wellknown NFS problems (security, performance,...).
- Data is transported to a client cache.
- To access a directory you need the appropr. entries in the access control list (ACL).
- AFS 4.0 is DCE/DFS.

AFS is a global, world wide filesystem. AFS does not have the wellknown problems of NFS, for example security or performance in a large distributed environment.

Data is transported by request from the AFS server to any AFS client into the AFS client cache.

To access files in a directory you need the appropriate entries in the directory access control list (ACL).

By the way, AFS 4.0 is the coming DCE/DFS, the distributed file system in the distributed computing environment.

Requirements to the backup of user data

- Users should be able to restore any of their files by themselves.
  - ➔ Incremental ADSM backup of user files
- The system administrator should be able to restore large sets of files in case of a disk failure.
  - ➔ Backup of complete user home directories ("buta")

The main requirement for the backup of user data is, that users should be able to restore any of their files by themselves. So we use the incremental ADSM backup of user files.

Because the system administrator should be able to restore large sets of files in case of a disk failure, we also backup complete user home directories with the AFS to ADSM API ("buta").

Problem 1: Large filesystems

- Large filesystems:
  /afs/urz.uni-heidelberg.de/usr/
  inst1/user1/
  user2/
  inst2/user1/
  user2/
- Currently 165 institutes and about 10.000 users.
- Database performance

The global filesystem AFS is - even for the own AFS cell - a very large filesystem. In the URZ Heidelberg the user directory contains the Heidelberg University institutes (currently 165). The user home directories are located below the institute directories. There are currently about 10.000 user home directories.

One can simply use the user directory as a virtual mountpoint for the user data backup. Restoring files using the ADSM GUI “by directory tree” is impossible. Database performance does not allow the usage of large filesystems.

(Our) Solution 1:

- Homedirectory of each user is an own ADSM filesystem. (VIRTUALMOUNTPOINT)
- dsm.sys and dsm.opt is modified when adsm client starts.
- Would be nice to have: VIRTUALMOUNTPOINT $HOME

Our solution to Problem 1 is to split the user directory in smaller pieces: The homedirectory of each user is an own ADSM filesystem by using the VIRTUALMOUNTPOINT option.

The configuration files “dsm.sys” and “dsm.opt”, which are just links to temporary files, are built when the front/end to dsm(c) is startet.

The modification of the two configuration files could be avoided, if one could specify a “VIRTUALMOUNT-POINT $HOME” option.

Problem 2: Large number of filesystems

- 10.000 user home directories = 10.000 filesystems
- ADSM GUI fills server window with 10.000 filesystems (about 6 MB data transferred to ADSM GUI, 15 minutes wait time).
- Database performance

Our solution to Problem 1 splits the large user directory into lots of smaller filesystems. This leads to another problem: The ADSM GUI fills the server window with all server known filesystems. For 10.000 filesystems about 6 MByte on data is transferred from the ADSM server to the ADSM GUI. This takes about 15 minutes time, which makes the ADSM GUI unusable.

Many filesystems for one ADSM client node decreases the database performance significantly.

(Our) Solution 2:

- Use multiple (virtual) ADSM clients with less filesystems, map user/institutes to (virtual) ADSM clients.
- Would be nice to have:
  Select display of server filesystems.

Our solution to Problem 2 is to split the large amount of filesystems into small pieces and map (groups of) institutes to different ADSM clients. Those clients can even run on the same UNIX workstation (virtual ADSM clients) by using the “-server=srv_name” and specifying a “NODENAME” option for each server entry in the dsm.sys configuration file.

Nevertheless it would be very nice, if we could restrict the display of the server filesystems for our users to the user home directory.

Problem 3: Many workstations with the same filesystems

- Users are distributed over many terminal servers.
- AFS always shows them their data.
- ADSM should also show them their data.

UNIX pool users are distributed over many terminal servers. AFS always shows the same user data to the users, regardless of the terminal server the user is working on.

This should also be true for ADSM: regardless of changing the UNIX workstation the user should always see the same ADSM backed up AFS data.

(Our) Solution 3:

- Use of the same ADSM client nodes on every workstation results to password problems (same encrypted ADSM password file on every workstation). So:
- Use R-commands (rsh, rcmd, ...) to run the ADSM client on the appropriate unix host.
- Would be nice to have:
  Better ADSM cluster support

Using the same ADSM client nodes on every workstation results to password problems, because the same encrypted ADSM password file has to reside on every workstation.

So our solution to Problem 3 is to use R-commands (rsh, rcmd, ...) to start the ADSM client on the appropriate unix workstation.

A better ADSM cluster support would reduce this problem.

Problem 4: Backing up and restoring AFS ACLs

- We use read authorization for a special backup userid for all user data.
- AFS ACLs are backed up correctly. They do not reside in the database, but on storage volumes.
- ADSM restores some directories, sets the ACLs and ... stops (no write access).

To back up the AFS user data to ADSM a special backup userid must have at least read access to the user data. We can justify a read access to our users, they cannot understand a write access.

AFS ACLs are backed up correctly. They do not reside in the ADSM database, but on storage volumes (cartridges).

Restoring complete directory trees shows, that ADSM restores part of the directory tree, sets then the AFS ACLs (read access) and then stops because of the missing write access.

(Our) Solution 4:

- Use AFS to ADSM API (“buta”) to restore most of the directory structure.
- Would be nice to have:
  Restore complete directory tree including all files first before setting the AFS ACLs.
  or
  RESTORE -DIRSONLY

Our solution to Problem 4 is to restore the latest user homedirectory backup by the AFS to ADSM API ("buta"). Almost all of the directory structure is then available. We then can recursively set the required write access. With ADSM Version 2 all missing files up to the latest ADSM incremental backup can be restored.

It would be nice, if ADSM would restore the complete directory tree including all files first before setting the AFS ACLs.

Nevertheless it would be nice, if we would have a "RESTORE -DIRSONLY" command, allowing to set the required write permission prior ro restoring the files themselves.

Conclusion:

- ADSM can be run in an AFS environment, if you use own front/end shell scripts and modify a couple of things.
- An AFS-ADSM backup strategy should use the combination of “buta” and ADSM.
- There are still some requests to the ADSM developers.

ADSM can be run in an AFS environment, if you use own front/end shell scripts and modify a couple of things.

An AFS to ADSM backup strategy should use the combination of a complete user homedirectory backup ("buta") and the incremental ADSM backup.

There are still some wishes and requests to the ADSM developers.