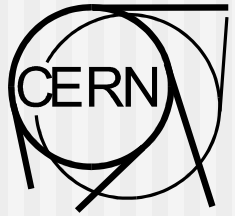
AFS Administration Framework

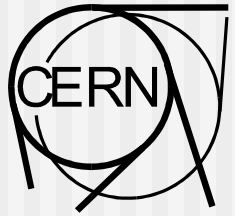
Wolfgang Friebel,
April 18.2002

AFS administration framework



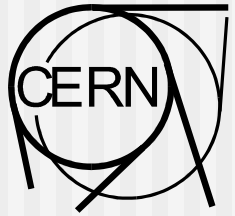
- Goals
- Current situation at CERN
- The new approach
- AFS Volume characteristics
- Description of volume collections
- afsadmin.cf (sample text)
- New features
- Backward compatibility
- Present status
- Internals
- Sample code
- Next steps

Goals



- Create a site independent framework for AFS administration
  - Reuse as much available software as possible
  - Modular design, building blocks can be replaced
  - Encapsulation, only the required interface is visible
- Create administration tools that use this framework (proof of concept)
- Migrate (some of) the existing administration tools in use at CERN to the new scheme

Current situation at CERN

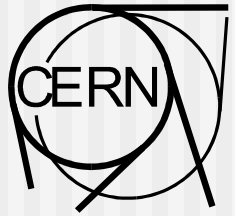


- Lots of tools in place, working environment
  - arc, a very generic tool to provide authenticated access
  - Structures within AFS to provide different services (home directories, software distribution, project space)
  - CERN specific code in many places
  - Complicated interdependency of tools
  - different pieces of code to solve the same (similar) task
- Some functionality missing from present tools

  Adding functionality would enforce interdependency

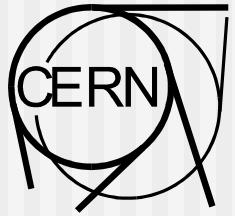
  Limitations in the present scheme hard to overcome

The new approach (1)



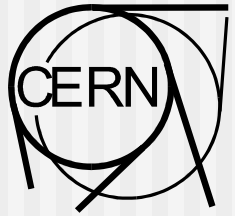
- Make use of existing perl module AFS.pm
  - not found at CPAN (does not fit into name space)
  - latest version at www.mpa-garching.mpg.de/~nog
- Provide an additional module Vos.pm
  - not all AFS library functions accessible from AFS.pm
  - especially missing: volume server access
  - main task: parsing of the volume and partition info
- Provide a framework to classify volumes
  - define collections of volumes with common features
  - Volset.pm is a package to handle such volume sets

The new approach (2)



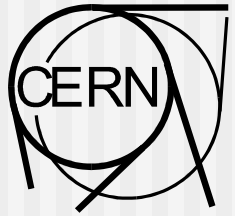
- Use the principles of object orientation
  - Separation of interface from internal workflow
  - Separation of application and tools name space
  - Modularisation and code reuse
- Keep as much as possible
  - Authentication based on arc
  - Code mostly in perl as before
  - Keep interfaces whereever possible
    - therefore procedural perl interface used

Grouping of AFS Volumes



- Old scheme: Pools, Projects, Users
  - central user vols handling (homedirs), no substructure
  - project volumes recognized by its name and mount point could be handled by project admins (delegation)
  - pools describe a group of AFS partitions where volumes with certain characteristics (e.g. backed up) are stored
- New scheme: Volume sets (Volsets)
  - each volume belongs to one or more volsets
  - There is a hierarchy of volsets, where more specific ones inherit features from the more general ones
  - the most generic volset (old scheme: pool) for a volume defines the affinity to partitions
  - The most specific volset describes quota, maintainer etc. (was project, the new scheme allows for subprojects as well)

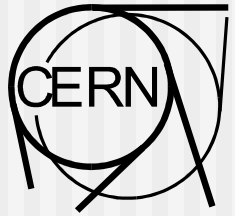
Description of Volsets



- Pools were described by pieces of perl code that needed to be "required". Some characteristics were even hardcoded.
- Projects were mostly described by the config file pv.cf which was used both by arc and some applications
- The new volsets are described in a single file afsadmin.cf

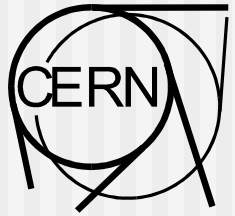
  It contains different sections describing the whole AFS space
  - The name of the AFS cell and the rule for the $HOME path
  - The available AFS servers and its functionality
  - A description of the available AFS partitions and the names of volsets that may be stored there (pools).
  - A list of volume patterns that define volsets
  - A list of volset names and its characteristics (mount point, quota, ...)
- The ACL's to administer volsets are kept on the arc server

New features (basics)



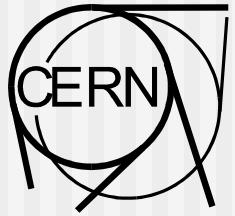
- Code is mostly free of site specific parts
- Role of afs servers needs not be hardcoded
- Pools and any number of projects or subprojects can be tied to certain AFS server partitions
- Scheme not restricted to projects, could be used to manage users volumes, ASIS etc. as well.

New features (details)



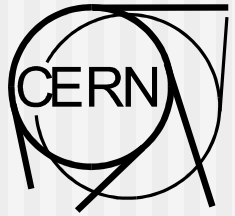
- Patterns use variable syntax to get a collection name from a volume name, much smaller number of patterns
- The collections form a hierarchy (going from left to right in the patterns section), child collections inherit from their ancestors (quota, partition,...) unless they provide more specific values
- Access to data (e.g. Quota information) is possible by functions only (separate namespace)
  - Changing the implementation will not change the interface
  - Functions internal to the modules are hidden

afsadmin.cf (sample text)



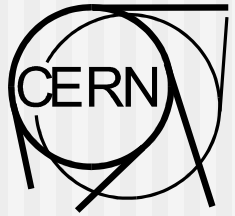

```
[AFSSERVERS]                        #server_name OS server_functions
afs78  Linux   fs
afsdb3 Solaris fs db arc reg
[PARTITIONS]                        #partition_names collection_names
afs78/(a,b,c)  fixed
afsdb3/s       sys
[VOLUMEPATTERNS]                    #perl_pattern collection_names
\.R$           recover
^s\.(\w+)\b    fixed $1 $1_s
^user\b        users
[VOLSETS]
ceres   4G exp/ceres                #collection_name quota mount_point
ceres_s 2G
```

Present status



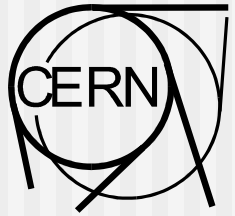
- Vos.pm is coded and tested, testsuite (testvos) is available (mostly parsing vos exa &co. output)
- Volset.pm coded and tested, testsuite existing, functionality mainly oriented towards needs of afs_admin tool to delegate pools and projects management
- Stubs for a generic mechanism to cache data in both perl modules available (user can influence behavior)
- Documentation available (perldoc Vos/Volset)
- Several programs rewritten to test new scheme
  - Fewer lines of code, faster
- Procedure to maintain project space at CERN (afs_admin) rewritten, provides enhanced functionality (first users in the Atlas and CMS experiments)

Internals



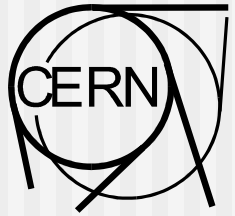
- Perl modules written with object orientation in mind. Presently a procedural interface is offered. Change to OO is simple (name space separation)
- All information internally held in structures (anonymous perl hashes which can contain substructures (anon hashes or anon arrays)
- Most data come with time stamps (for caching)
- Access to data from outside exclusively by functions (encapsulation)

Sample code



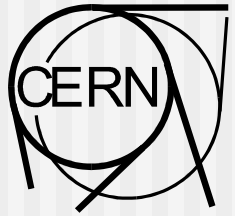

```
# $vol can be volname or volID
 my $volname=vid2name($vol);
# retrieve some volume attribute
 my $quota=quota($volname);
 my $timestamp=get_volattrib($volname, '_time');
# get all volsets the volume is in ($volsets[0] is poolname)
 my @volsets=vol2set($volname);
# create a volume (without sanity checks)
 my $to=choose_disk($volname);
# $to contains suitable partition, e.g. filesrv1/a
 $to =~ s/\// /;
 arc_execute("vos create $to $volname");
```

Next steps



- Try to integrate other tools into afs_admin
  - Code to create or delete projects
  - Code to add or delete administrators for a volset
- Rewrite all the tools that still use the old scheme
- Other tools could also make use of the framework, rewrite/revisit them as well?
- Advantages would be
  - no site dependency of tools
  - Take advantage of cached information
  - Less repetitions of code (e.g. parsing the VLDB format)
- Make the framework publicly available (combination with other efforts, e.g. AFS.pm)

Availability



- Vos.pm and Volset.pm are on ftp://ftp.ifh.de/pub/unix/gnu/perl/modules (Vos-1.04.tar.gz and Volset-1.04.tar.gz)
- AFS.pm can be found there as well or at http://www.mpa-garching.mpg.de/~nog
- afs_admin (requires a running arc daemon) and the server parts of it (arc procedures) are available on request

  (email to Wolfgang.Friebel@cern.ch)