CHIMERA - A NEW, FAST, EXTENSIBLE AND GRID ENABLED NAMESPACE SERVICE

Mr. Mkrtchyan Tigran, Dr. Fuhrmann Patrick, Mr. Gasthuber Martin
DESY, Hamburg, Germany

Abstract

After successful implementation and deployment of the dCache system over the last years, one of the additional required services, the *namespace* service, has faced additional and completely new requirements. Most of them are caused by the scaling of the system, the integration with Grid services and the need for redundant (high availability) configurations. The existing system, having only NFSv2 access path, is easy to understand and is well accepted by the users. This single 'access path' limits data management task of making use of classical tools like 'find', 'ls' and others. This is intuitive for most users, but failed while dealing with millions of entries (files) and more sophisticated organisational schemes (metadata). The new system should support a native programmable interface (deeply coupled, yet fast), a 'classical' NFS path (now version 3 at least), a dCache native access and an SQL path allowing any type of metadata to be used in complex queries. Extensions with other 'access paths' will be possible. Based on the experience with the current system we highlight on the following requirements:

- large file support (64 Bit) + large number of files (>10^8)
- fast
- platform independence (runtime + persistent objects)
- Grid name service integration
- custom dCache integration
- redundant, highly available runtime configurations (concurrent backup etc.)
- user accessible metadata (store and query)
- ACL support
- pluggable authentication (e.g. GSSAPI)
- external processes can register for namespace events (e.g. removal/creation of files)

A detailed analysis of the requirements, the chosen design and selection of existing components will be discussed. The current schedule should allow to show the first running prototype – a RDBMS back ended file system with NFSv3 interface and alternative (JDBC) access path to files metadata.

THE PROJECT GOAL

Modern experiments produce terabytes of data, which has to be managed by tape storage systems. While the user-intuitive way to access data is through file names, the storage systems normally deal with tapes, offsets and disks. A system, which could have a file system view from one side and interact with storage system from the other side became crucial. Based on our experience and actual needs a list of requirements was compiled:

- **Unique file ID independent from name**
  File names are not persistent, while data is. We can rename files, but still be able to access original data;
- **Name-to-ID and vice versa mapping**
  By referencing files in storage system by ID we need a possibility to find the file ID while users will operate by file names;
- **Callback on file system events, like remove and move**
  Removing a file in the file system has to trigger an associated action of file removal in the storage system. Moving a file from one directory to another may trigger a migration of the file from one storage system to another;
- **Directory tags, inherited by subdirectories**
  possibility to define default values, like OSM-group or file-family dependence, or to which tape-set a file have to reside in. Usually, directories are created prior to files, and de-facto become a natural holder of initial values;
- **Metadata association with files**
  arbitrary metadata can be associated with files, in particular storage system specific information like tape name, offset and so on;
- **Worm holes**
  A convenient feature: files that are not shown in the directory listing, but are available in all directories. Can be used for distributing configuration files;
- **Additional channel for the client to access metadata**
  client applications have to be able to store and retrieve metadata.

CURRENT SOLUTION

In 1997, we have introduced PNFS[1] – an *NFS server* on top of a database. PNFS allows all NFSv2 operations except actual data IO. The data access is performed by native store/retrieve utilities of the storage system. The implementation is based on user-space *NFS daemon*, which communicates with the *DB-server* through a shared-memory block. The *DB-server* simulates a file system on top of gdbm. Each subdirectory can have its own *DB-server*, which runs as a separate process. Access to metadata is done through special file name syntax.

Currently there are two HEP labs that heavily rely on PNFS – DESY and FNAL, and few others that use PNFS as a component of *dCache* in LCG2. At DESY we have 55 *DB-server* processes, serving more than 3 million file

entries, which corresponds to 500TB of data in HSM with 1KHz access rate. All databases together uses 20GB disk space.

PNFS is being used by various storage systems – *Enstore[2], OSM*, *dCache[3]*. *Enstore* and *OSM* store references to files – "bit file IDs", which are used by HSM to identify files. *dCache* stores file locations, e.g. pool names. In the past some experiment-specific file access libraries used to store file locations in SHIFT pools, now replaced by *dCache*.

Despite successful deployment of PNFS, we found spots which may cause limitations in future.

- Max. file size 2 GB due to NFSv2 specification
- Metadata access only through NFS:
  - → no direct path for attached storage systems;
  - → all metadata types use the same channel and the store:
    - heavy access to metadata by storage system has performance impacts on regular NFS operations;
- Metadata are stored as BLOB:
  - → no metadata query functionality;
- No ACLs
- NFS security (= no security), although we can disable some NFS operations (remove)

NEW IMPLEMENTATION

While the file size limitation is solved by new NFSv3 front-end, metadata access path needs changes in design. Since we heavily depend on metadata stored in PNFS, a high throughput access to metadata becomes crucial for very large installations. In the mean time, the main "customer", dCache, was modified to optionally store instance metadata, *cacheinfo*, in private database. We consider two possible solutions: using a file system with DMAPI[5] (Data management API) support, like JFS or XFS, or simulating a file system on top of RDBMS. Each approach has it's own advantages and disadvantages:

Table 1: RDBMS evaluation

Advantages of RDBMS	Disadvantages of RDBMS
Query Language Automatic database partitioning Backup Consistency check Triggers Stored procedures JDBC/ODBC makes implementation independent	Difficult to put file system tree into tables Performance with growing number of clients and entries not investigated.

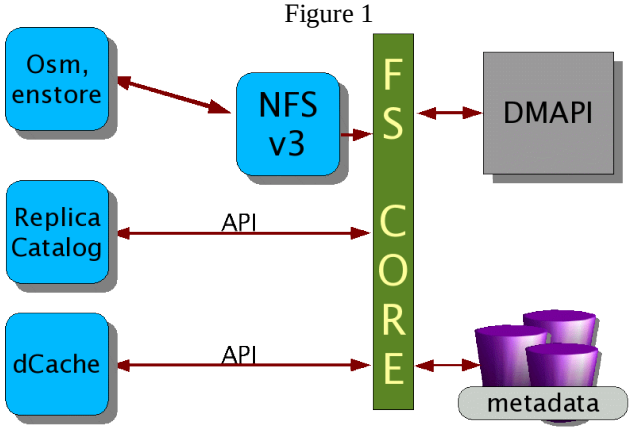
Table 2: DMAPI evaluation

Advantages of DMAPI	Disadvantages of DMAPI
Well known Vendor support Existing implementations for SGI, Linux, Solaris Existing backup tools Data Management API Posix ACL's Can be shared by any known protocol	Still metadata and file names in the same location No directory tag inheritance. No wormholes with standard sharing protocols (NFS)

Possibly, a combination of both approaches will be taken.

The original idea of implementing a GRID Replica Catalogue as a core component was prohibited. The Replica Catalogue interface will become one of the external access interfaces (Figure 1).

Figure 1



It's obvious that nowadays the UNIX permissions are insufficient for many applications, especially in GRID context. Most of modern file systems support ACLs. The choice here is not obvious either:

- NT ACL's
- POSIX ACLs (many drafts, no actual standard)
  Posix 1003.6 draft 13;
  Posix 1003.1e draft 15;
- UNIX Variants
  Based on various Posix drafts, with some extensions;
- DCE (AFS) ACLs
  based on draft 13 with a fair number of extensions;
- GRID-map file
  More or less UNIX-like – readers/writers;

In addition POSIX(UNIX) and NT ACLs have a different behaviour:

Posix – uid/gid based, first/best match
NT – SID (principal) based, order independent

(Posix draft 13 corresponds to a subset of NT ACL's.)

While most of HEP applications are UNIX-based, we have seen growing demand of GRID-based access, where user DN(Distinct Name) replaces the uid. Currently, our strategy moves in direction of NT ACL's, at least the subset used by POSIX plus principal handling, but it's still under discussions.

CONCLUSIONS AND OUTLOOK

During the decades of deployment PNFS has done a decent job being stable, robust and flexible. To keep that high standard also in the future it needs some modifications and additions. The NFSv3 front-end exists and is currently in the test phase. A running prototype of an RDBMS based file system simulation is available and currently being tested in terms of performance and scalability. We are in contact with DMAPI-enabled filesystem developers to check out all needed functionality in DMAPI and underlying filesystems. During design evaluation, GRID Replica Catalogue concept shifted from core functionality area to optional access interface. The situation with ACLs is not fully clear yet and we are in contact and discussions with other developers. Although we are familiar with *enstore*, *osm* and *dCache*, a provided solution will help to manage other storage systems as well.

Investigations are needed to choose the most appropriate solution and, like mythological chimera*, we need to involve several technologies to achieve our goal.

REFERENCES

[1] http://www-pnfs.desy.de/
[2] http://hppc.fnal.gov/enstore/index.html
[3] http://www.dcache.org
[4] http://www.ietf.org/rfc/rfc1813.txt
[5] http://www.opengroup.org/pubs/catalog/c429.htm

* An animal from ancient Greek mythology with a lion's head and fore parts, a goat's body, a dragon's rear, and a tail in the form of a snake.