Chimera - a new, fast, extensible and Grid enabled namespace service

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

Abstract

After successfully implementing and deploying the dCache system over the last years, one of the core 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, using the NFSv2 access path only, is easy to understand and is well accepted by the users. It's intuitive for most users, but failed when 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 focus 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, high 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)

THE PROJECT GOAL

Modern experiments produce terabytes of data, which have to be managed by tape storage systems. While the user-intuitive way to access data is through filenames, storage systems normally deal with tapes, offsets and disks. A system, which could have a filesystem view from one side and interacts with storage systems 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**
  Filenames are not persistent, while data is. We can rename files, but still are able to access the original data;
- **Name-to-ID and vice versa mapping**
  By referencing files in the storage system by ID we need a possibility to find the file ID while users will operate on filenames;
- **Callback on filesystem evens, like remove and move**
  Removing a file in the filesystem 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**
  A possibility to define default values, like OSM-group or file-family, or on which tape-set a file has to reside. 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[4] operations except actual data IO. The data access is performed by native store/retrieve utilities of the storage system. The implementation is based on a user-space *NFS daemon*, which communicates with the *DB-server* through a shared-memory block. The *DB-server* simulates a filesystem on top of a gdbm database . Each subdirectory can have its own *DB-server,* running as a separate process. Access to metadata is done through a special file name syntax.

Currently there are two HEP labs heavily relying on PNFS – DESY and FNAL, and many others using PNFS as a component of *dCache* in LCG2. At DESY 61 *DB-server* processes serving more than 3 million file entries, which corresponds to 500TB of data in the HSM with a 1KHz access rate. All databases together occupy 20GB of 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 the 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*.

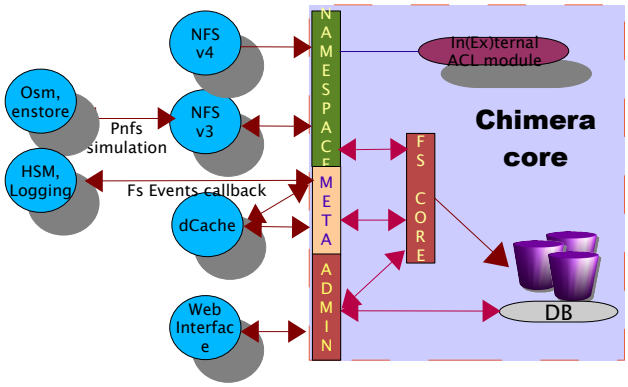
Despite successful deployment of PNFS, we found spots which may cause limitations in future.

- 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 a new NFSv3[5] front-end, the 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. After some design evaluations we decided to simulate a filesystem on top of RDBMS.

Pic. 1



As an option we looked at DMAPI enabled file systems like IBM™ JFS or SGI™ XFS. As there wer no out of the box solution, vendor dependencies, lack of big users community and still required coding, we decided not to go in this direction.

The basic idea of a new design is a filesystem core on top of a relational database. A relational database provides the following benefits:

- Well known (world wide expertise)
- Query language
- Backup and recovery tools
- Consistency checks
- Triggers and stored procedures
- JDBC/ODBC allows to be vendor independent

By providing a separate database table for each type of metadata (pnfs level), we could isolate levels so that queries on one type of information don't have any impact on others. In addition, this schema doesn't store empty records if a file does not have metadata of this particular type. Simple SQL queries can be used to get all kinds of information (for example space used by particular user/VO which is necessary for the upcoming quota integration).

Due to talking strict JDBC Chimera is database vendor independent. The current prototype has been tested against Oracle, PostgreSQL and MySQL RDBMS. External configuration files allow to define DB implementation specific SQL dialects to improve performance.

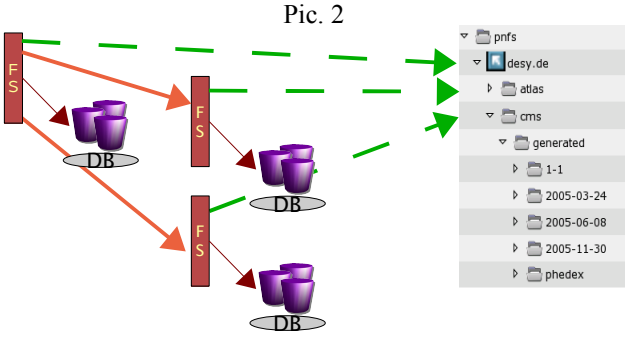
A well defined API provides three sets of operations – namespace operations, metadata manipulations and admin interface. The dCache system accesses the filesystem directly via the API, bypassing the NFS interface. By using the Chimera implementation of the Java *File* class it's easy to add a customized frontend. Nevertheless an NFS interface is required to support legacy clients. In addition, the NFS interface has to simulate at least a subset of the pnfs *magic* commands. Internal transaction isolation allows to run many frontends (NFS, dCache and so on) against a single database (filesystem) in parallel.

Nowadays the UNIX permissions are insufficient for many applications. While most HEP applications are UNIX-based, we have seen a growing demand of GRID-based access, where user DN(Distinct Name) replaces the uid. Pluggable permission handlers allow to add a site specific security policy implementation. If nothing is defined, the UNIX standard permission handling is built in.

In preparation of the new namespace service dCache module, which interacts with pnfs, the PnfsManager had to be changed. The monolithic concept was replaced by a modular architecture - core controller with three functional modules for namespace, storage and location operations. Each module is an external unit. One module may provide functionality of other modules as well. Chimera provides namespace and storage operations. File locations in pools, e.g. *cacheinfo*, is provided by the dCache instance itself, know as '*Companion*', because it's related to a particular instance.

Like a regular UNIX file system, which allows to mount new hard disk to a directory, Chimera allows to set a directory as reference to another instance. This doesn't only improve scalability, but provides a hierarchical structure as well, where each braunch can be the *root* of a different 'view'. E.g.: H1, Zeus and the LHC experiments may have their own instances, although they are still part of a single rooted file system, representing the whole DESY storage system.

Pic. 2



The event handling system provides callback functionality to external applications. This allows performing actions on file system events (for instance: remove file from HSM on removal from namespace).

The test instance at DESY easily handles 1.5 million files with a 200 files per second create rate. In combination with dCache components we are running a prototype for evaluation. The pnfs simulation allows to use existing clients, like dccp and osmcp, without any code changes. Nevertheless further stress test are required.

To enable a smooth migration of existing pnfs installations we are going to provide a set of tools, which will be available with the first official release.

We are still in discussions with other mass storage system experts to make the HSM interface and the storage information as generic as possible.

CONCLUSIONS AND OUTLOOK

During one decade of deployment PNFS, has proven to be stable, robust and flexible. To keep that high standard also in the future we need to provide some modifications and additions. The working prototype with the corresponding dCache components is available and under performance, scalability and stability tests. A full functional release for beta-testers is expected in June 2006. The official release is expected in the first quarter of 2007. Chimera has been designed and optimized for dCache interactions. Nevertheless the service is independent of the dCache software and may be used as filesystem namespace provider for other applications as well. As soon as Chimera moves from the development to production phase we are aiming to make it available to the community (LGPL License).

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/rfc1094.txt
[5] http://www.ietf.org/rfc/rfc1813.txt