

Enstore with Chimera namespace provider

Dmitry Litvintsev, Alexander Moibenko, Gene Oleynik, Michael Zalokar

Fermi National Accelerator Laboratory, P.O. Box 500, Batavia, IL 60510-5011, USA

E-mail: {litvinse,moibenko,oleynik,zalokar}@fnal.gov

Abstract. Enstore is a mass storage system developed by Fermilab that provides distributed access and management of the data stored on tapes. It uses namespace service, pnfs, developed by DESY to provide filesystem-like view of the stored data. Pnfs is a legacy product and is being replaced by a new implementation, called Chimera, which is also developed by DESY. The Chimera namespace offers multiple advantages over the pnfs in terms of performance and functionality. Enstore client component, encp, has been modified to work with Chimera or any other namespace provider. We performed high load end-to-end acceptance test of Enstore with Chimera namespace. This paper describes modifications to Enstore, test procedure and results of the acceptance testing.

1. Introduction

Enstore is a Mass Storage System developed and operated by Fermilab. It allows client applications distributed across IP networks to seamlessly access data stored on permanent media [1].

The system provides hierarchical view of stored files via NFS mounted file system, served by namespace provider, the PNFS (a Perfectly Normal File System), developed by DESY [2]. The PNFS has been designed and implemented in the end of 90s and has reached its limitations. In the run up to LHC data challenge DESY has developed next generation namespace provider called Chimera [3] that meets scalability and performance requirements of the LHC experiments.

This article describes recent migration of Enstore system from PNFS to Chimera namespace provider.

2. Enstore

The Enstore project was started in 1998, in preparation for the data taking in Run II of the Fermilab Tevatron collider. It has delivered a highly reliable, scalable, fault tolerant, and flexible storage system that satisfied the needs of the two major Run II experiments, D0 and CDF, as well as a host of smaller experiments. It has also been used to address general mass storage needs like system and user data backups. Currently, Enstore is the primary storage system at Fermilab. It holds about 40 TiBs of data amounting to 50 million files. It can be operated stand-alone or can use dCache [4] or SAM [5] systems, as disk caching front-ends for optimal tape I/O.

Enstore is a client-server application implemented mostly in Python with inclusion of C codes for modules where high performance is required. The server is a multicomponent structure of distributed servers that provides:

Enstore at Fermilab

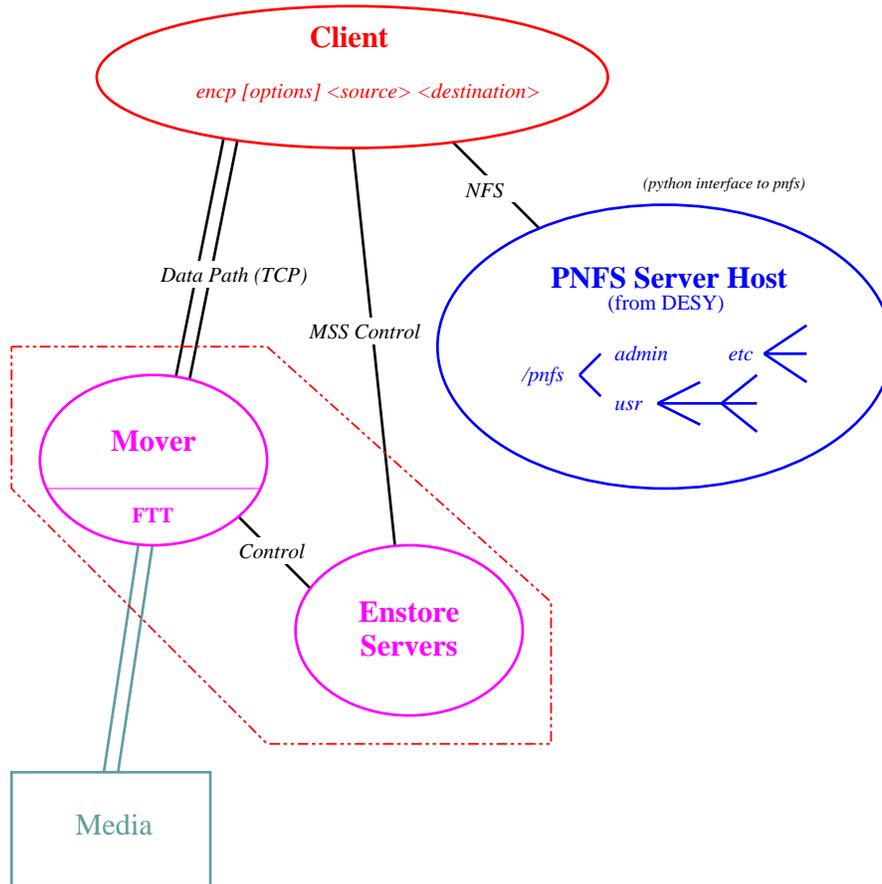


Figure 1. Schematics of encp client interaction with Enstore system.

- hierarchical meta-data view of user files stored on tape, presented to client as if it were a Unix file system;
- management of user files (e.g. renaming, removal or recovery);
- distributed access to tape drives;
- interface to robotic tape libraries;
- resource management of available tape drives;
- flow control based on meta-data associated with files.
- tape allocation accounting per storage group, file family, media type;
- self-monitoring, error-reporting and alarm services;
- periodic data integrity checks;
- web-interface;
- file migration from older media type to new media type (e.g. LTO3 → T10KC);
- small file aggregation.

Enstore client, encp, provides “cp”-like functionality to retrieve/store files from/to tapes, end-to-end data integrity verification and meta-data manipulation.

Enstore components run on distributed commodity hardware connected via IP networks and communicate using UDP-based inter-process communications. Great care has been taken to ensure that the system will function well under extreme load conditions. By design, there is no preset limit on the number of concurrent user computers or on the number of physical media libraries or drives. The system is only limited by the availability of physical resources.

The system is designed for distributed and peer-to-peer reliability. Each request originating from the encp is branded with a unique ID. Encp attempts retries under well-defined circumstances, issuing an equivalent request with a new unique ID. The system can instruct encp to retry if it needs to back out of an operation.

A simplified interaction between the encp and the Enstore system is presented in figure 1. The file hierarchy is presented to the client via an NFS mounted filesystem served by the PNFS. A client command to write a file into Enstore has the form:

```
encp [options] <source> <destination>
```

where <source> is file name and <destination> is the full path name or a directory name in PNFS. When this command is invoked, encp starts by checking destination for write permissions using the PNFS interface. It also extracts flow control meta-data stored in special tag files located in the destination directory which contain the name of the storage group, library name, file family, file family width, and file family wrapper. Once this is done, encp communicates with the configuration server to discover the address of the corresponding library manager (LM). It then sends source file and flow control information to the LM, which, in turn, queries the volume clerk (VC) for available tape that satisfies selection criteria based on file size, storage group name, file family name, and file family wrapper name. As soon as tape is available, the LM schedules a request in the queue. The LM begins servicing this request when a mover becomes available for the tape by sending the mover's IP address to the encp client. The encp client then establishes a TCP connection directly with the mover to initiate data transfer to the mover which, in turn, writes the data to the tape. Upon completion of the transfer, the encp client updates the file metadata at the destination and sets additional metadata in special layer files which includes a unique file identifier (BFID), volume label, location of file on tape, size and crc values, fild id (pnfsid), file name, and tape drive information.

Reading of files from Enstore proceeds in a similar fashion. It begins with checking PNFS for read permissions, extracting file layer information, and passing this information to the corresponding LM. The LM then schedules the request in its queue which is serviced when a mover becomes available. The available mover first mounts the tape in case it is not yet mounted, and, as soon as the encp client receives the mover's IP address, data is transferred via TCP to the client host.

3. Namespace provider

The namespace provider is the software component that provides the association between the file names and the data distributed across multiple locations and on a variety of media.

It supplies:

- Unique file ID independent of file name.
- Path to ID and reverse ID to path mapping.
- File meta-data storage. Arbitrary meta-data may be associated with the file, in particular storage system specific information like tape volume name, offset etc.
- Directory tags inherited by sub directories to store extra meta-data associated with storage system like storage group, file family and so on.
- Callbacks on filesystem events.

- Worm holes. A convenient feature: files that are not shown in directory listing, but are available in all directories. Useful for distributing configuration files.
- Additional channel for the client to access meta-data.

When the Enstore project was begun over a decade ago, PNFS was chosen as its namespace provider service. PNFS, which is an NFS server running on top of a database, allows all NFSv2 operations except actual data I/O which is performed by storage system specific utilities. It is implemented as an NFS daemon running in user space and communicating with a db-server through a shared memory block. The db-server simulates a filesystem on top of a database (PostgreSQL or GDBM back-ends are supported).

While PNFS has been used successfully during Run II of the Fermilab Tevatron Collider, serving about 10M files each to D0 and CDF, the following limitations have been identified:

- Maximum file size is 2 GB due to NFSv2 specification.
- Meta-data access only through NFS server making it a bottleneck. Heavy access to meta-data by storage system adversely impacts NFS operations.
- Meta-data are stored as (key,value) pair with key being unique file id and value being BLOB. This impacts performance (e.g. for ls operations) and does not allow for selective meta-data querying.
- No ACL support.
- No security.

As the LHC experiments anticipated a substantial increase in data rates, requiring higher throughput rates and better scalability in terms of the performance and capacity of storage systems, with additional requirements for ACL support and authentication, it became clear that a new namespace provider was needed.

In 2004, DESY developed the next generation namespace provider, Chimera, in order to address the needs of the LHC. Chimera is a filesystem simulator running on top of a relational database management system (RDBMS). It is designed for performance and provides a well defined API for namespace operations, metadata manipulations, and administration. Storage systems can access the filesystem directly via the API, eliminating the need for the NFS interface used in PNFS.

Chimera is written in Java and is therefore highly portable. It uses JDBC interface for database access without data base specific bindings. This allows to deploy Chimera on top of any database that has JDBC driver. A relational database, depending on implementation, provides the benefits of a query language, consistency checks, triggers and stored procedures, and backup and recovery tools. The database schema is designed to allow metadata query isolation by providing a table for each type of metadata, so that queries for one type of metadata do not impact those for others.

Chimera supports ACLs via pluggable permission handlers with standard Unix handlers as default and GSS authentication. Most importantly, Chimera implements the NFSv4.1 protocol for namespace and data file access which allows parallel POSIX I/O on distributed data. The NFSv4.1 implementation has been production ready since dCache version 1.9.12.

4. Chimera and Enstore

Improvements in performance and scalability demonstrated by Chimera, its extended feature set, and its support of NFSv4.1 prompted Enstore developers to evaluate Chimera as a namespace provider for Enstore.

Technically, the only namespace aware component of Enstore is the encp client. With PNFS, Enstore used the extended filesystem interface to handle file metadata. This interface has now been made into a base class with two concrete subclasses `Pnfs` or `Chimera`. A specific interface is

instantiated at run time based on system specific tags available in NFS served filesystems, using the factory method pattern. Chimera support has been available starting with encp version v3_10e.

5. Acceptance Test

The goal of the acceptance test was to determine whether Enstore with the Chimera namespace provider would be stable under heavy loads similar to those found in an actual production environment. Test involved performing encp transfers directly to/from Enstore as well as dcp transfers via dCache for end-to-end testing.

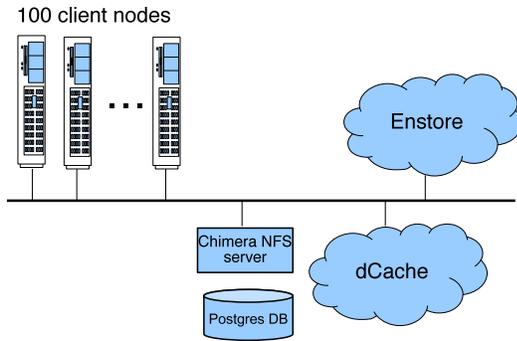


Figure 2. Enstore with Chimera acceptance test setup.

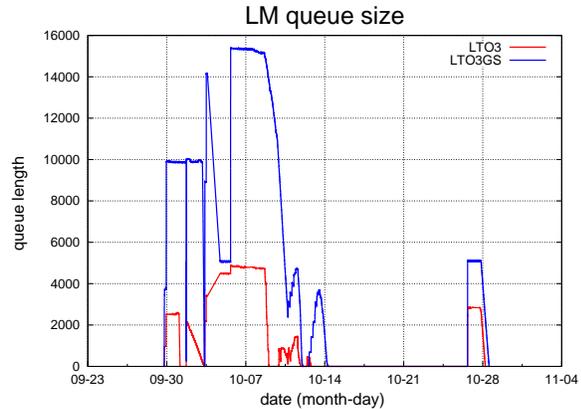


Figure 3. Library Manager queues during acceptance test. Two LMs were involved in the test.

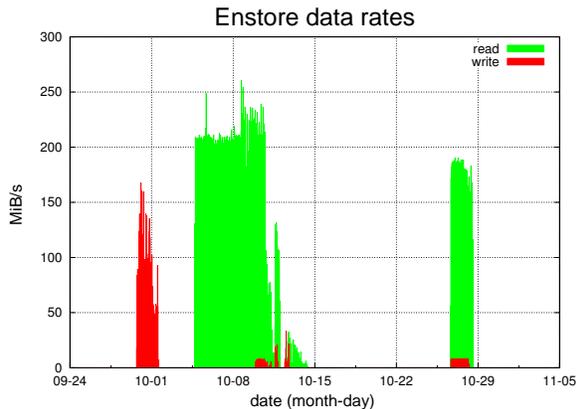


Figure 4. Enstore data rates.

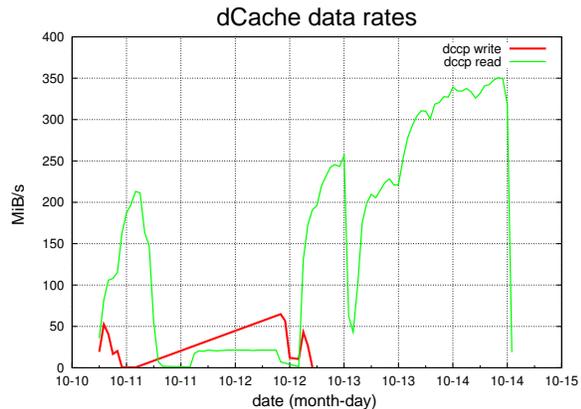


Figure 5. dCache data rates.

The acceptance test setup is shown in figure 2, it consisted of:

- 100 client nodes.
- Fully functional Enstore instance attached to SL8500 robotic library. The system had:
 - Separate namespace node to run Chimera, underlying PostgreSQL data base and dCache PnfsManager.
 - Separate node for Enstore internal databases and database servers.

- Two LTO3 library managers (LMs) running on separated nodes.
- 5 LTO3 movers.
- dCache system attached to Enstore :
 - 10 pool nodes. One dcap door per node.
 - Separate head node.
 - dCache version 1.9.5-28 (support for Enstore and Chimera is available in dCache starting version 1.9.5-24).

Each client node ran a torture test harness in a loop, submitting about 130 simultaneous encp/dccp file transfers. Files were generated to contain random binary data with an average size of 2 GiB. First, write tests were performed until all tapes were used up. Then the read tests commenced. A portion of the tapes was later recycled and dccp write tests were performed for smaller file sizes.

The load, created on the system, is expressed in terms of LM queue sizes as shown in the plot in figure 3. As can be seen, the LM sustained up to 15K requests in the queue without dropping requests. This far exceeds typical queue sizes, such as those found on CMS T1 production systems, where the peak load is about 5K-10K requests.

Table 1. Enstore Chimera tests summary

transfers	TiBs	Enstore/dCache	r/w
83198	161.5	Enstore	read
23749	25.4	Enstore	write
1215	2.3	dCache	read
9869	1.4	dCache	write

The data transfer rates, in MiB/s, seen by the Enstore and dCache systems are shown in figures 4 and 5, respectively. Enstore rates were limited by the number of available movers while dCache rates were limited by network and aggregate disk I/O. Table 1 gives a summary of the transfers. No errors were seen and Chimera performed flawlessly.

The data transfer rates in MiB/s seen by Enstore and dCache system are shown in figures 4 and 5 correspondingly. Enstore rates were limited by the number of available movers while dCache rates were limited by network and aggregate disk I/O. Table 1 gives summary of transfers. No errors have been seen. Chimera performed flawlessly.

6. PNFS → Chimera migration

Fermilab operates three Enstore instances which include public Enstore, D0 Enstore, and CDF Enstore. CMS T1 uses public Enstore but employs a separate PNFS namespace. There are, therefore, four PNFS systems at Fermilab.

The PNFS to Chimera migration procedure is outlined in figure 6. It involves several steps that have been wrapped into an all-or-nothing Python script and the migration procedure has been tested several times on copies of PNFS databases taken from all three production Enstore systems.

The migration procedure starts with setting up a Chimera PostgreSQL database, stopping PNFS, starting the Chimera server, creating top level directories, and mapping their "Chimera" IDs to corresponding PNFS IDs. Then, the Chimera server is stopped, the PNFS server is started, and the `pnfsDump` script is run. After the `pnfsDump` script completes, three sets of files

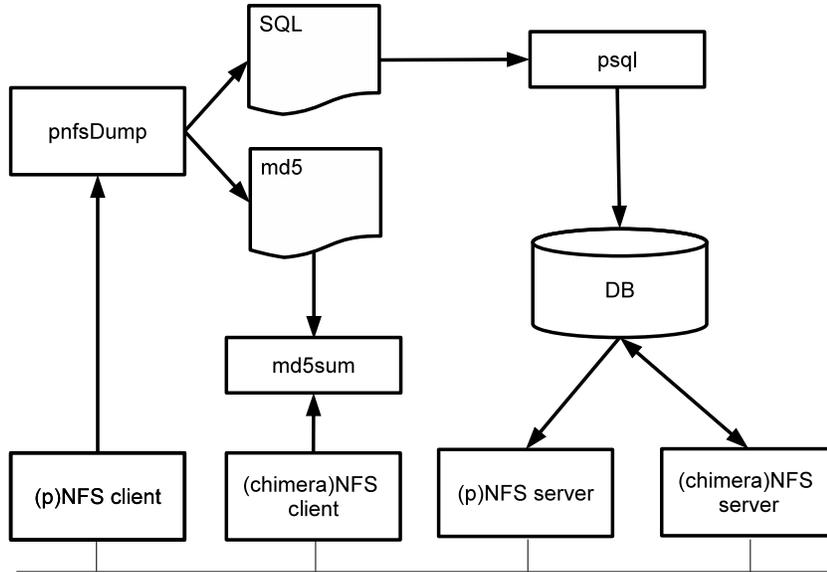


Figure 6. Diagram of PNFS to Chimera migration procedure.

are produced for each top level directory: list of SQL statements to be injected into the Chimera database to create the directory tree structure underneath, list of all files in the directory tree, and the file containing the list of md5 checksums for each file. At the next step, the migration script takes the SQL statement lists and injects them into the Chimera database using a `psql` client.

After the previous step completes, the file metadata stored in layer 4 is converted into Chimera location information. The companion information, which contains a list of dCache pools where the files are located, is also converted to Chimera location information. Then the PNFS server is stopped and the Chimera server started. At this point, the system is available. In the background, the script executes `md5sum` on all files in the Chimera filesystem and compares results against md5 checksum lists produced by `pnfsDump`. The time consuming part is `pnfsDump` and the SQL injection. To minimize effects of the downtime, the PNFS server remains open for users in read-only mode (by setting the PNFS database connection user to a read-only role).

We performed migration of the public Enstore system’s namespace server from PNFS to Chimera on 02/23/12 and simultaneously migrated PNFS to Chimera on D0 and CDF systems on 05/01/12. No issues were encountered during migration or in the subsequent running of production Enstore instances with the Chimera back-end. The migration proceeded smoothly and the transition was absolutely transparent to the users. No changes on the users’ ends were necessary beyond upgrading `encp` to version `v3_10e`. This step had been completed well ahead of the actual migration due to the fact that this version of `encp` provided simultaneous support for PNFS and Chimera.

Timing for various stages of the PNFS to Chimera migration is summarized in table 2. The CDF Enstore system was the fastest to migrate while the D0 system was the slowest. Comparing the number of files and directories presented in Table 2, it can be concluded that this was due to the larger number of files per directories in the D0 system. Companion import was not done for D0 because D0 uses a SAM instead of dCache as a front-end.

Table 2. PNFS to Chimera migration timing

action	public Enstore	CDF	D0
pnfsDump	6h52m	5h32m	12h53m
SQL Import	9h47m	5h38m	8h28m
layer4 to location info	1h08m	0h26m	2h00m
companion import	0h17m	0h20m	N/A
Total	18h4m	11h56m	23h21m
number of files	15317841	8523362	14001375
number of dirs	323053	686947	19468

7. Conclusion

After almost a decade of using the PNFS namespace provider, the Fermilab production Enstore systems have now been switched to the new Chimera namespace provider. Migration from PNFS to Chimera proceeded smoothly and operational experience with the new system is being gained. So far, no problems have been observed. In the future, we plan to fully explore the capabilities of the NFSv4.1 interface, supported by Chimera, to provide POSIX I/O for stored data to the Fermilab user community.

References

- [1] Bakken J, Berman E, Huang C, Moibenko A, Petravick D, Rechenmacher R, Ruthmansdorfer K “Enstore Technical Design Document” URL <http://www.dcache.org>
- [2] Fuhrmann P, “A Perfectly Normal File System PNFS” <http://www-pnfs.desy.de/>
- [3] Gasthuber M, Mkrtchyan T, Fuhrmann P 2005 “Chimera - a new, fast, extensible and Grid enabled namespace service” *Proc. Conf. Computing in high energy physics and nuclear physics September 27-October 1 2004 (Interlaken)* p 1180
- [4] dCache URL <http://www.dcache.org>
- [5] Lueking L *et al.* 1998 “The Sequential access model for Run II data management” *Proc. Conf. Computing in high energy physics and nuclear physics September 1-4 1998 (Chicago) (FERMILAB-CONF-98-382-E)*