dCache, a distributed storage data caching system

Michael Ernst, Patrick Fuhrmann, Martin Gasthuber, Tigran Mkrtchyan[1], Charles Waldman[2]
[1](DESY, IT)
[2](FERMI, CD-INTEGRATED SYSTEMS)

Abstract

This article is about a piece of middle ware, allowing to convert a dump tape based Tertiary Storage System into a multi petabyte random access device with thousands of channels. Using typical caching mechanisms, the software optimizes the access to the underlying Storage System and makes better use of possibly expensive drives and robots or allows to integrate cheap and slow devices without introducing unacceptable performance degradation. In addition, using the standard NFS2 protocol, the dCache provides a unique view into the storage repository, hiding the physical location of the file data, cached or tape only. Bulk data transfer is supported through the kerberized FTP protocol and a C-API, providing the posix file access semantics. Dataset staging and disk space management is performed invisibly to the data clients.
The project is a DESY, Fermilab joint effort to overcome limitations in the usage of tertiary storage resources common to many HEP labs.
The distributed cache nodes may range from high performance SGI machines to commodity CERN Linux-IDE like file server models. Different cache nodes are assumed to have different affinities to particular storage groups or file sets. Affinities may be defined manually or are calculated by the dCache based on topology considerations. Cache nodes may have different disk space management policies to match the large variety of applications from raw data to user analysis data pools.

Keywords: Storage Manager, Cache System, Java, nfs, security

1 Scopes

1.1 Scope of this document

This document presents the ideas, status and results of a software product, developed by a joint effort between the DESY IT and the FERMI CD devisions. The paper is intended to create curiosity and not to describe the system in full detail. Therefore we picked a small subset of features which may be regarded as the signature of our product. Moreover, this document will not prove that Hierarchical Storage Systems benefits from well known optimization methods like **Rate Adaption, Deferred Write, Staging and Read Ahead**. It is assumed that the reader consults the results of those studies at [6].

1.2 Scope of the product

Although the initial goal of the project was to create a compact system enabling a small set of HSMs[1] to work more efficient, we learned[5] that by using modern technologies and by adding reasonable effort we would be able to cover a spectrum of features which would make the system a generic tool for caching, storing and easily accessing huge amounts of data, distributed among a large set of heterogeneous caching nodes.

The product, subsequently called the **dCache**, is a standalone system providing a file name space distributer and a data pool manager, being able to coordinate a large amount of caching nodes. The system may run without an HSM backend or with an arbitrary number of different HSM classes and/or instances. To optimize the HSM access, the *dCache* uses methods like *Rate Adaption, Deferred Write and Staging.* The *Read Ahead* approach is in preparation.

[1]Hierarchical Storage Manager. This term is used within this document to denote Storage Systems base on the IEEE[1] standard

To distribute files among multiple caching nodes, different attraction schemes may be used. The system is able to detect data hot spots and, if required, distributes the most frequently used datasets among multiple nodes. This feature allows that, although we provide a single unique namespace, a cached file may reside within more than one pool. Besides FTP and a private random access protocol (**dCap**[2]), we can easily integrate other methods even without restarting the cache system. This and and other convenient features are enabled by using the modern programming language *Java*[7][5]. The online configuration is presented to the administrator through a command line or web interface. Both require strong authentication and support fine grained access control lists.

2 Step by Step

2.1 Product Placement

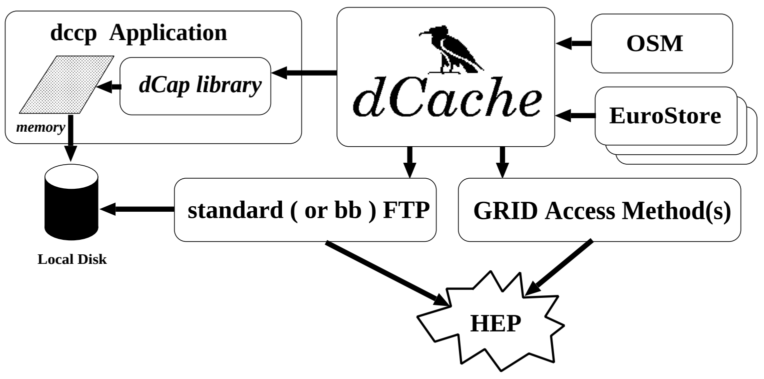

Figure 1: dCache Placement

The *dCache* system is located between a regular HSM[3] and the actual end user application. We currently support FTP clients (regular or kerberized) and a dCache private random access protocol, **dCap**.

2.2 The HSM backend

The interface between the HSM and the dCache System is highly configurable and sufficiently flexible to support different instances of the same HSM and/or even different HSMs within the same dCache instance. Moreover the dCache may run without backend HSM and as such can be used as a large scalable file store.

By using standard staging/caching methods like *Rate Adaption, Deferred Write and Staging*, the dCache allows to make better use of the connected HSM or to integrate slower and possibly less expensive equipment without overall performance penalty.

2.3 The Pool Affinity Scheme

A single dCache instance is able to control an arbitrary number of so called **Cache Pools**. Cache Pools are responsible for fetching files from the connected HSMs, if required by a reading client, as well as grouping incoming files and flush them to one of the HSMs as soon as a timeout has expired or space is running short. In addition, Pools launch movers talking the appropriate protocol to send/receive data to/from the external clients.

[2]dCache Access Protocol

A set of criteria determines which of the available Cache Pools is assumed to hold the required dataset. The first three items described below are statically configured while the last item makes use of real time status information taken from the caching pools to achieve load balancing among them.

Backend HSM criteria Most HSMs support some kind of internal grouping mechanism which mainly predicts the tapeset a file will be written to or read from. (Storage Group for OSM[1] and file family for Enstore[4]). The dCache supports these groups transiently and allows to assign them to Cache Pools together with read and write preferences. This feature allows to have dedicated Cache Pools for physics working groups. Different preferences enforce overflow pools in case space is running short for a particular working group.

Filesystem Topology In addition to the grouping mapped to the HSM groups, pools can be configured to have an affinity to specific file system directories or directory trees. This is necessary to support dCache instances without HSM backend systems.

IP Topology The IP number or subnet of the requesting client can be used as well to determine the responsible Cache Pool. This allows to have Cache Pools per subnet for network load balancing.

Cost Efficiency (Space and Load) In case more than one cache pool could be assigned to hold the data, the relevant pools are asked for a cost claim about storing the file. This cost is evaluated by taking the space and the CPU load of the pools into account.

2.4 Single Filename Space and Hot Spot Spreading

The dCache gives a unique view of its content together with the content of the backend storage system with the help of an NFS server[3] (pnfs[2][8]). This allows the standard unix file name manipulation tools (*rm, mv, ls ..*) to be used. The position or the name of an entry within this namespace is independent of the location of the actual dataset. It may reside on

- a cache disk only.
- within the HSM only.
- within the HSM plus somewhere on at least one cache disk.

The distribution of hot spot datasets among different cache pools is done transiently. The NFS namespace and therefore the user doesn't notice this replication at all.

2.5 Access methods, Security and Redundancy

Within the dCache model, access methods are abstracted by means of pluggins. Therefore it's quite easy to implement new methods if required. They don't influence the dCache kernel at all. These entry points are called *Doors* in the dCache context and we are currently supporting the FTP and the dCap door. Each type of door has its mover counterpart which talks the corresponding data protocol. Doors have well known addresses while the data connection is coming directly from the pools holding the request file or, for write accesses, is willing to accept the incoming files.

The level of security is totally determined by the implementation of the door. Fermilab had a strong requirement for a kerberized FTP door due to their *Strong Authentication Project.* At DESY, we have to follow a smooth migration path to allow our experiments to keep track. We will run the dCap protocol with the security level of the NFS2 protocol (nearly nothing), but only within the DESY intranet. Accesses from outside for both protocols, dCap and FTP, have to present a valid Kerberos5 ticket. On the client side, the dCap protocol is implemented in a c-library, containing the posix like dc_open, dc_read, dc_write, dc_seek and dc_close I/O

[3]Currently we are using the NFS2 protocol, but an upgrade has already started

functions and for convenience we provide a *dccp* application which is essentially the *cp* command linked against the dCap c-library. Within DESY this library assumes to find a mounted PNFS filesystem which reflects the dCache file name space. In general the library accepts a http like syntax (*dCap://pnfs/desy.de/...*) to access files within our repository without a mounted filesystem. The *Door Concept* allows to have additional Door instances running on different hosts to allow load sharing and fail overs in case of server problems. Within DESY the PNFS filesystem informs the dCap library which door to use. From outside DESY we could use alias names and smart name server redirects to chose the server, but we don't yet.

2.6 Administration Interfaces and Security

As well as the data access doors there are doors talking the ssh protocol and allow administrative tasks to be done. The standard *ssh* client can be used as well as a graphical interface to login. Both methods have the ssh security level which we regard as sufficient for all our purposes. Administrative actions are protected by Access Control Lists which are checked against the authenticated user.

3 Results and Status

We are running the development system since April and we have requested several millions of files from Linux, Solaris and Irix servers. In the meantime we don't have any doubts concerning the stability. The performance using the dCap c-library has been evaluated by the teams from the experiments and turned out to be comparable with the rfio performance. Due to these positive results the experiments already integrated the dCap library into their data chain. The majority of the experiments agreed to convert their own caching systems into dCache before end of 2001.

4 Acknowledgment

We want to thank our partners Jon A. Bakken, Igor Mandrichenko and Donald Petravick at FERMI for their competent input. Due to the facts that this project spans all DESY experiments and that we invited them to use the system from the beginning we need to thank our integration teams for their patience and superb support and feedback. Eduard Avetisyan, Alan Campbell, Serguei Essenov, Ulrich Fricke, Dmitri Goloubkov, Rainer Mankel, Dmitri Ozerov, Armine Rostomyanand, Krzysztof Wrona and Marek Kowal for the initial version of the client c-library. And, not to forget, Christine Prehm for taking care of everything.

References

[1] The Open Storage Manager, Lachman Technology Inc.
[2] Patrick Fuhrmann, A perfectly normal namespace for the DESY Open Storage Manager, presented at Conf. on Computing in High Energy Physics, Berlin, 1997.
[3] IEEE P1244, Reference Model for Open Storage Systems Interconnection - Mass Storage System Reference Model Version 5, September 1994.
[4] Enstore, The Fermi Data Storage System, `http://hppc.fnal.gov/enstore`
[5] Andy Kowalski, JASMine - Jefferson Lab Asynchronous Storage Manager `http://cc.jlab.org/scicomp/JASMine`
[6] dCache, White Paper `http://www-dcache.desy.de/dCacheDesign.html`.
[7] Bruce Eckel, Thinking in Java 2, Prentice Hall, `www.phprt.com`.
[8] Patrick Fuhrmann, A Perfectly Normal FileSystem,`http://watphrakeo.desy.de/pnfs`