dCache on Steroids - Delegated Storage Solutions

T Mkrtchyan¹, F Adeyemi¹, A Ashish¹, G Behrmann², P Fuhrmann¹, D Litvintsev³, P Millar¹, A Rossi³, M Sahakyan¹ and J Starek¹

¹Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany
²Nordic e-Infrastructure Collaboration (NeIC), Copenhagen, Denmark
³Fermi National Accelerator Laboratory, Batavia, USA

E-mail: tigran.mkrtchyan@desy.de

Abstract

For over a decade, dCache.org has delivered a robust software used at more than 80 Universities and research institutes around the world, allowing these sites to provide reliable storage services for the WLCG experiments as well as many other scientific communities. The flexible architecture of dCache allows running it in a wide variety of configurations and platforms - from a SoC based all-in-one Raspberry-Pi up to hundreds of nodes in a multi-petabyte installation. Due to lack of managed storage at the time, dCache implemented data placement, replication and data integrity directly. Today, many alternatives are available: S3, GlusterFS, CEPH and others. While such solutions position themselves as scalable storage systems, they cannot be used by many scientific communities out of the box. The absence of community-accepted authentication and authorization mechanisms, the use of product specific protocols and the lack of namespace are some of the reasons that prevent wide-scale adoption of these alternatives. Most of these limitations are already solved by dCache. By delegating low-level storage management functionality to the above-mentioned new systems and providing the missing layer through dCache, we provide a solution which combines the benefits of both worlds - industry standard storage building blocks with the access protocols and authentication required by scientific communities. In this paper, we focus on CEPH, a popular software for clustered storage that supports file, block and object interfaces. CEPH is often used in modern computing centers, for example as a backend to OpenStack services. We will show prototypes of dCache running with a CEPH backend and discuss the benefits and limitations of such an approach. We will also outline the roadmap for supporting ‘delegated storage’ within the dCache releases.

1. Introduction

dCache[1][2] is a software solution that provides a highly scalable, distributed storage system with a single-rooted namespace by combining a number of individual nodes. By adding more storage nodes, the total capacity and throughput are increased simultaneously. dCache has several mechanisms for data placement and protection: file replication for redundancy, hopping manager to migrate data from WAN ‘input buffers’ to LAN ‘local servers’, in order to separate incoming streaming data from random access from the local analysis farms, for example, pool-to-pool replication on load, in order to offload transfers from hot data servers and various migration tasks, to adjust data placement or evacuate/retire one of the data servers. In addition to storage management, dCache supports multiple access protocols for storing and retrieving data: DCAP, GridFTP, HTTP, NFS, WebDAV and XrootD. Industry standard as well as HEP-specific authentication and authorization mechanisms are provided: Kerberos, OpenID-Connect, Username+Password, X509.
As described above, dCache functionality can be split into three categories:

- Aggregating multiple storage nodes into a single rooted distributed storage system.
- Providing access to the storage system via various access protocols and security mechanisms.
- Managing data placement, replication and reliability.

2. Motivation

For decades, hardware/software RAID-based[3] systems were used by dCache as storage nodes. With the growing size of a single hard disk as well as the number of disks in a single box, this approach does not scale any more. A single disk failure requires immediate action to replace the failed disk. This adds an operational overhead and increases the total cost of ownership (TCO). Moreover, the long RAID rebuild times, – about 1TB per hour, – decrease overall node performance and reduce end-user experience.

A storage node, which is tolerant to multiple disk failures with a low operational overhead, is required to provide a reliable service.

3. CEPH

There are multiple distributed storage systems available today. Unfortunately, they cannot fulfill all the requirements of scientific communities - either due to the lack or incompatibility of the usual authentication mechanisms or specific access protocols. Nevertheless, by combining such systems with dCache we can provide the usual functionality and still benefit from functionality provided by systems like CEPH[4].

CEPH is an open-source storage platform which implements a single distributed object store. It operates without a single point of failure and is scalable to the exabyte level. Data protection is achieved by replication and erasure coding. As a result of its design, the system is both self-healing and self-managing, minimizing administration time and other costs.

Because of its capabilities, CEPH is often used as a backend storage for OpenStack installations and is already deployed at a number of sites, which makes it attractive as a backend for dCache as well.

In CEPH, a minimal storage unit called OSD – object storage device, – is typically a single disk. One or more OSDs are grouped into so-called placement groups (PG). Storage of an object in a placement group triggers the creation of the required number of replicas on one or more OSDs. A virtual container, or pool, consists of one or more placement groups (Figure 1). The replication factor and/or erasure coding rules are attached to the pool. The “Controlled Replication Under Scalable Hashing” algorithm, or CRUSH[5], is used for selecting an appropriate placement group when creating or accessing an object.
4. dCache - CEPH integration

CEPH provides object-, block-, and file interfaces. The object interface provides basic PUT, GET and DELETE operations. While storing data in CEPH as objects is a common practice, it does not come without its limitations. Specifically, the object interface is less suitable for integration with dCache because the objects are stored on a single OSD, making it impossible to write them using random access, which is required by the dcap, nfs and xrootd protocols. On the other hand, the block interface allows random access on so-called rados block devices (RBD). Moreover, RBD supports striping over multiple OSDs and provides read-through/write-back caching out of the box, which makes the block interface very attractive for dCache integration. For every file stored in the dCache a new RBD image is created. Note, however, that this is not a commonly used practice for CEPH; hence large scale performance studies are required.

In dCache, the component which stores data is also called a pool (which may create confusion in the CEPH context). All pools operate independently from each other and make their own decisions when to flush files to tape, when to remove cached copies and when to destroy the removed files. An isolated design of this sort makes it very easy to scale up dCache instances to thousands of pools without performance degradation caused by inter-communication overhead. A pool’s repository has two main functions: tracking the state of the stored files, – such as being received from client, being received from store, or needing to be flushed to tape, – and providing IO access to the data. In addition to tracking file states, the pool stores a subset of metadata associated with a file, such as file size, access latency and retention policy.
remains unchanged, but any IO operations are sent directly to CEPH via librados. This approach keeps other pool components unaware of CEPH backend, while preserving the pool’s full functionality, including migration, p2p and HSM connectivity (Figure 2).

5. HSM connectivity
To flush or restore a file to/from HSM, dCache uses an external script. Upon flushing this script takes the local path of the file to be flushed and returns an HSM-specific URI, which is stored by dCache. Upon restore, the script takes the previously stored URI and the local path to where the file should be restored:

```
hsm.sh put /local/path/to/file
```

or

```
hsm.sh get -uri <uri> /local/path/to/file
```

When the pool uses CEPH as a backend, then a CEPH-specific URI is used instead of the local path:

```
ceph-hsm.sh get -uri <uri> rbd://<ceph_pool>/<file_id>
```

The `file_id` corresponds to the file’s unique identifier within dCache, called the `pnfsid`.

6. Current status
Starting from dCache version 3.0.0, a single configuration option can be used to leverage CEPH as a backend for the pools:

```
pool.backend = ceph
```

Additionally, the location of `ceph.conf` and the cluster name can be configured.

As with any other functionality in dCache, CEPH support passed all our testing. However, since none of the collaborating sites such as DESY, Fermilab and NDGF, are using it in production, an intensive pre-production testing by sites is essential.

7. Performance evaluation
The development process was concentrated on stability, functional feature set and interoperability of CEPH and POSIX based pools. Additionally, any performance evaluation require deep knowledge and production experience of CEPH. This hard task we leave to site admins who was waiting for dCache-CEPH integration.

8. Future plans
As mentioned earlier, when using CEPH as a backend, each dCache pool is associated with a single CEPH pool. While this naturally fits with the current dCache design, it limits the functionality which CEPH or any other shared backend can provide. As any node in the CEPH cluster can access any
CEPH pool, a new paradigm can be created where a single CEPH pool can be served by multiple dCache pools. This turns dCache pools into a thin layer, acting as a stateless gateway between the namespace, the various protocols and the backend storage. They can be dynamically started and stopped as more bandwidth is required.

It makes sense to push replication and data migration down into the underlying storage system. However, with the growing demand for distributed installations, we cannot assume that all dCache pools will be running on top of CEPH within a single instance. As a result, some installations will contain both CEPH-based and regular RAID-based areas. Such deployments still must be able to move data between different pools, independent of their backend type (Figure 3).

![Figure 3. CEPH and regular pool coexists in the same deployment](image)

9. Summary
Due to operational overhead, traditional RAID-based systems become less suitable as building blocks for large-scale storage systems. Nowadays there are plenty of alternative technologies available which aim to provide richer functionality and flexibility without compromising stability. Effective version 3.0.0, dCache can use CEPH as a backend for data storage and still provide a full range of functionality including replication, migration and HSM connectivity. Based on production experience, more of the native CEPH functionality will be directly used by dCache. Nevertheless, dCache will continue to support traditional RAID-based systems and even support mixed environments, where CEPH- and RAID-backed pools can be mixed in a single installation.

References
[1] www.dcache.org
[2] dCache, the Overview https://www.dcache.org/manuals/dcache-whitepaper-light.pdf
[3] Redundant Array of Independent Disks https://en.wikipedia.org/wiki/RAID
[4] www.ceph.com
[5] The CRUSH algorithm http://docs.ceph.com/docs/master/rados/operations/crush-map/