Xrootd in dCache - design and experiences

Gerd Behrmann¹, Dmitry Ozerov² and Thomas Zangerl³
¹Kastruplundsgade 22, DK-2770 Kastrup, Denmark ²Notkestraße 85, D-22607 Hamburg, Germany ³Teknikringen 14, S-10044 Stockholm, Sweden
E-mail: behrmann@ndgf.org

Abstract. dCache is a well established distributed storage solution used in both high energy physics computing and other disciplines. An overview of the implementation of the xrootd data access protocol within dCache is presented. The performance of various access mechanisms is studied and compared and it is concluded that our implementation is as performant as other protocols. This makes dCache a compelling alternative to the Scalla software suite implementation of xrootd, with added value from broad protocol support, including the IETF approved NFS 4.1 protocol.

1. Introduction

Physics experiments such as those conducted at CERN produce both large amounts of data and have specific requirements about performance properties of data access protocols. Modern industry-standard network protocols such as NFS 4.1¹ provide data access with high-performance and low-latency, but they are not yet universally available in network stacks of operating systems and client applications. For this and for other reasons, xrootd ² has become an increasingly popular data access protocol among particle physicists and beyond.

dCache³ is a well established distributed storage solution used in both high energy physics computing and other disciplines. A cornerstone of dCache is the orthogonality of name space, data storage, and access protocols. This allows dCache to support a wide array of local and wide area data access and management protocols including (Grid)FTP, HTTP(S), WebDAV, SRM, NFS 4.1, and more.

The xrootd protocol is the native data access protocol of the Scalla software suite⁴. One of the goals of the Scalla software suite is to provide low latency high bandwidth data access. The Scalla software suite is used in many high energy physics projects as well. This implementation provides network request handling, protocol message processing and a translation layer to native file systems. For consolidated namespace management the cluster management service daemon (cmsd) is required.

For many computing centers processing physics analysis data, the combination of the two above daemons will not be sufficient; support for further network protocols may be needed or a better namespace management or compatibility with specific physical storage systems or other features that only specialised storage management software may offer.

Frequent solutions to this issue contain running xrootd/cmsd on top of the storage provider (e.g. HDFS⁵) or integrating the native xrootd protocol as a plugin (Castor⁶, DPM⁷). As a plugin to the storage provider, however, it might bypass the storage provider’s load balancing mechanisms and thereby render them overall ineffective. It may also be difficult for the storage...
provider to maintain uniform logging and billing output or consistently structured configuration options.

Therefore, dCache offers xrootd in a native Java-based implementation as one of its supported storage access network protocols. As namespace, physical storage and network access constitute orthogonal functionalities in dCache, xrootd can be used along a number of complementary access protocol such as WebDAV or NFS4.1 in combination with different storage backends without much adaptation.

2. Comparison of local access protocols in dCache

It may be argued that dCache’s native access protocol, DCAP, makes supporting xrootd in dCache unnecessary. Indeed, the two protocols are quite similar, supporting a wide range of meta data operations, prestaging, vector read, asynchronous IO, and more.

The main motivation for supporting xrootd in addition to DCAP is client support. The ROOT Data Analysis Framework[8] is widely used in the high energy physics community. Although DCAP is available as a plugin to ROOT, xrootd is natively supported by ROOT and is thus more widely available. Additionally, xrootd is currently being considered as a wide area access protocol within the Worldwide LHC Computing Grid (WLCG).

Even for clients that support both protocols, experience has shown that the same job may have quite different IO access patterns depending on which protocol is used. This is not caused by difference in the protocol, but by difference in the client implementation. Detailed analysis of an Atlas job used during our empirical evaluation in section [4] demonstrates this difference: With DCAP the job read 10 times as much data as with xrootd (4 GB vs 400 MB), however xrootd submitted approximately 8 times as many read requests (32500 for DCAP vs 250,000 for xrootd). With DCAP the job issued an average request size of 128 kb vs 1712 bytes for xrootd. The size of the file used by the job was 1 GB. It is important to stress that these figures are for the same computational job. The differences are caused by tradeoffs in the client library: the DCAP client library uses aggressive read ahead and issues 128kb reads. The xrootd client library on the other hand uses an internal cache to avoid reading the same request twice.

With such radical differences in the access profile, large differences in job efficiency are expected. Section [4] shows that each protocol is faster in some scenarios and thus there is reason to support both in dCache.

3. Implementation of xrootd in dCache

A native Java implementation of xrootd has been included in releases of dCache for a long time, but more recent releases have focused on a performant and feature-complete reimplementation.

3.1. Redirector node and data server

Conceptually, the xrootd concept of a redirector node and a data server has been mapped to dCache’s key architectural components of doors and movers. Doors constitute the dCache endpoint that is normally first contacted by a client for a given protocol. Responsibilities of dCache doors usually include authorization, contacting the namespace provider and sending transfer requests to the dCache system to initiate pool selection. Once a pool has been selected and the transfer (mover) is ready, the dCache door redirects the client to the pool with the mover, thus acting as an xrootd redirector node. The movers and pools take the role of the xrootd data server, whereby functionality provided by cmsd like MSS integration or a single namespace that spans across multiple servers is already included in specialized shared components of dCache.

3.2. Request flow

As xrootd in dCache is implemented natively in pure Java, protocol message processing, connection handling and authentication as well as authorization had to be rewritten according to
Both on dCache doors and pools, the JBoss Netty asynchronous event-driven network application framework\textsuperscript{[10]} is used as the main building block for network communication and connection handling. As the same request/response messages are issued to both the door and the pool, only one protocol handler had to be created and is employed on both the door and the pool.

Figure\textsuperscript{[1]} details the redirect process between door and pool for an open request. After the namespace lookup of the requested resource, the door generates a unique one time token which it passes on to the pool. Once the mover has started and communicated the network address of the pool to the door, the door replies with a redirect message to the client, including the one time token as opaque information.

This one time token is added to the request URL by the client when it issues the open request to the pool again, where the running xrootd mover selects the right transfer based on the token. From there on, the client and the pool exchange data directly.

Commands that only necessitate namespace operations, such as removal of files and directories, directory listing requests or copying and moving of files, are fulfilled directly by the door and will not result in a redirect to the pool.

3.3. Authentication and authorization

The Scalla xrootd implementation follows the notion of pluggable security mechanisms, where the appropriate plugins are selected as a result of the client-server message exchange.

dCache has long supported ALICE file-catalogue based token authorization\textsuperscript{[11]}. Additionally, the most recent version of dCache introduces support for authentication using the GSI variant implemented by xrootd clients. After authentication, the user information obtained by the authentication mechanism is sent to gPlazma\textsuperscript{[12]}, dCache’s authorization service. The response of this service constitutes the foundation for authorization decisions both for operations on the namespace as well as on the pool. Hence, the authorization configuration setup by site administrators on dCache instances also applies to the xrootd service and the authorization behavior is uniform across services.

As in the Scalla xrootd implementation, authentication is only performed on the door for performance reasons.

3.4. Performance and scalability considerations

Catering to the needs of the high energy physics community, the xrootd implementation in dCache had to satisfy both the requirements of being highly performant while scaling to a large number of simultaneous requests. While the JBoss Netty framework is already optimized for high performance and scalability\textsuperscript{[13]}, dCache’s xrootd implementation tries to improve the performance further by reading response blocks ahead, supporting vector read requests and by using Java NIO for disk I/O.

In redirect-based protocols, dCache has traditionally started transfers on the pool on a new TCP port and then redirected the client to that port. As the number of totally available TCP ports is limited, this had negative effects on the scalability of transfers. To increase the number of possible simultaneous transfers, a single xrootd server on a single TCP port handles client requests on the pool and selects the correct transfer based on the UUID included in the client’s open request. If no client is connected to the pool and no transfer is active, the server handling the client requests automatically shuts down, thus unblocking the port.

3.5. Implementation summary

In summary, the different layers of the Scalla xrootd protocol stack have been mapped to functionality already present in dCache, newly developed handlers and pre-existing libraries. Special care has been taken to end up with a performant and feature-complete product that
Figure 1. Interaction between the doors and pools with the xrootd implementation in dCache

Figure 2. Layers of the xrootd protocol stack in xrootd (left) and dCache (right)

takes the advantages of the protocol into account as well as possible. Figure 2 visualizes the components realizing the protocol stack in Scalla xrootd compared to dCache.

4. Empirical evaluation
Results obtained at the HEPiX Storage Working Group Laboratory (KIT) compare the efficiency of the xrootd implementation in dCache 1.9.10 with an older version in dCache 1.9.7, the Scalla implementation and the dcap protocol in dCache. The results are summarized in figure 3. While the older implementation of xrootd in dCache processes the least events for all numbers of jobs and dcap is in the middle field, Scalla xrootd and the xrootd in current dCache versions are approximately on par. Scalla xrootd has a slightly higher event throughput when a low number of jobs is executed and dCache xrootd manages to process more events if a larger number of jobs is utilized.

Recent HammerCloud tests performed at DESY Hamburg gave a further hint at such a trend. The test compares xrootd and dcap in dCache 1.9.10. The analysis used in this test is a typical ATLAS Athena Monte Carlo analysis analysing the AOD data format. The analysis is primarily selecting muons and muon trigger objects. The Athena version used to process

1 Details about the test setup can be found at [15].
Figure 3. Comparison of number of events processed by xrootd in different dCache versions, dcap and Scalla xrootd.

Figure 4. Comparison of HammerCloud results for dcap and xrootd in dCache 1.9.10

the AODs is 16.0.2.3 (ROOT version 5.26/00e). The Athena versions used to reconstruct and produce the AODs are 15.6.x (ROOT version 5.26/00e). The hardware setup is identical to the one described in [17]. The test results are summarized in figure 3. The results are consistent with those shown in figure 3. The xrootd implementation scales better than dcap, but has lower throughput under low load conditions.

The comparison between DCAP and xrootd however has to be seen in the light of the large differences in client access behaviour described in section 2. Aggressiv read-ahead in DCAP clients causes more data to be read while reducing disk seeks and round trips. In particular under low load when files fit in the file system cache of the disk pools and while network bandwidth is plentiful, DCAP has a performance advantage. With many concurrent jobs the read-ahead logic eventually saturates the bandwidth and at that point xrootd will perform better. The increased amount of data read by DCAP may also cause disk bandwidth to become an issue, in particular when many files are read concurrently and the files can no longer fit in the file system cache.
5. Conclusion and outlook

An overview of the implementation of xrootd in dCache was presented. It was shown that xrootd abstractions such as redirector and data servers fit well into the dCache architecture of protocol doors and data pools. Performance comparison with other protocols in dCache and other implementations of the xrootd protocol show that dCache’s xrootd implementation is performant.

The implementation of the xrootd protocol in dCache is fairly complete, with the notable absence of multiple concurrent TCP streams for data transfer. We have no plans to implement this part of the protocol in dCache.

Long term we see improvements in client code the key to increased efficiency. Most promising for local area access is the recently finalized NFS 4.1 protocol, as client code is embedded in the operating system. Applications can thus benefit from the file system cache layer and read-ahead logic in the operating system kernel rather than reimplementing this in the application layer. Like xrootd, NFS 4.1 fits well with the overall architecture of dCache and is fully supported in dCache 1.9.10 [17].

6. Acknowledgments

We would like to thank the HEPiX Storage Working Group and specially Andrei Maslennikov for providing their measurement data to us, Johannes Elmsheuser for the help with the HammerCloud tests, and Yves Kemp for helping with the DESY setup.

References

[1] S. Shepler, S. Inc, M. Eisler, and D. Noveck, “Network file system (NFS) version 4 minor version 1 protocol,” January 2010. http://www.ietf.org/rfc/rfc5661.txt.

[2] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky, “Xrootd - a highly scalable architecture for data access.”

[3] M. Ernst, P. Fuhrmann, T. Mkrtchyan, J. Bakken, I. Fisk, T. Perelmutov, and D. Petravick, “Managed data storage and data access services for data grids,” in Computing in High Energy and Nuclear Physics - CHEP 2004 (A. Aimar, J. Harvey, and N. Knoors, eds.), (Geneva), CERN, 2005.

[4] C. Boehme, A. Hanushevsky, D. Leith, R. Melen, R. Mount, T. Pulliam, and B. Weeks, “Scalla: Scalable cluster architecture for low latency access using xrootd and oibd servers.” http://xrootd.slac.stanford.edu/papers/Scalla-Intro.pdf.

[5] “Hadoop xrootd.” https://twiki.grid.iu.edu/bin/view/Storage/HadoopXrootd.

[6] http://castor.web.cern.ch/castor/

[7] R. Rocha, “Standard protocols in DPM,” in Computing in High Energy and Nuclear Physics - CHEP 2010, (Taipei), October 2010.

[8] R. Brun, F. Rademakers, and S. Panacek, “Root, an object oriented data analysis framework,” 2000.

[9] A. Hanushevsky, “The scalla-xrootd protocol version 2.9.6.” http://xrootd.slac.stanford.edu/doc/prod/Xrdv296.pdf July 2009.

[10] http://www.jboss.org/netty

[11] D. Feichtinger and A. J. Peters, “Authorization of data access in distributed storage systems,” in Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing, GRID ’05, (Washington, DC, USA), pp. 172–178, IEEE Computer Society, 2005.

[12] A. Rana, “gPLAZMA: Introducing RBAC security in dCache,” in Computing in High Energy and Nuclear Physics - CHEP 2006, 2006.

[13] http://gleamynode.net/articles/22232/

[14] http://w3.hepix.org/storage/storagewg.php.

[15] A. Maslennikov, “Hepix storage working group – progress report 1.2010,” April 2010. http://w3.hepix.org/storage/hep_pdf/2010/Spring/HaslennekovSWG_PROGRESS_Rep.pdf.

[16] D. C. van der Ster, J. Elmsheuser, M. U. Garcia, and M. Paladini, “HammerCloud: A stress testing system for distributed analysis,” in Computing in High Energy and Nuclear Physics - CHEP 2010, 2011.

[17] J. Elmsheuser, P. Fuhrmann, Y. Kemp, T. Mkrtchyan, D. Ozerov, and H. Stadie, “LHC data analysis using NFSv4.1 (pNFS): A detailed evaluation,” in Computing in High Energy and Nuclear Physics - CHEP 2010, 2011.