FERMILAB-CONF-17-407-CD
ACCEPTED

Accessing Data Federations with CVMFS

Derek Weitzel[1], Brian Bockelman[1], Dave Dykstra[2], Jakob Blomer[3], Ren Meusel[3]

[1] University of Nebraska - Lincoln Holland Computing Center, US
[2] Fermilab, Batavia, IL, US
[3] CERN, Geneva, CH

E-mail: dweitzel@cse.unl.edu

Abstract. Data federations have become an increasingly common tool for large collaborations such as CMS and Atlas to efficiently distribute large data files. Unfortunately, these typically are implemented with weak namespace semantics and a non-POSIX API. On the other hand, CVMFS has provided a POSIX-compliant read-only interface for use cases with a small working set size (such as software distribution). The metadata required for the CVMFS POSIX interface is distributed through a caching hierarchy, allowing it to scale to the level of about a hundred thousand hosts. In this paper, we will describe our contributions to CVMFS that merges the data scalability of XRootD-based data federations (such as AAA) with metadata scalability and POSIX interface of CVMFS. We modified CVMFS so it can serve unmodified files without copying them to the repository server. CVMFS 2.2.0 is also able to redirect requests for data files to servers outside of the CVMFS content distribution network. Finally, we added the ability to manage authorization and authentication using security credentials such as X509 proxy certificates. We combined these modifications with the OSGs StashCache regional XRootD caching infrastructure to create a cached data distribution network. We will show performance metrics accessing the data federation through CVMFS compared to direct data federation access. Additionally, we will discuss the improved user experience of providing access to a data federation through a POSIX filesystem.

1. Introduction

In [1], we defined a federated storage system to be a *collection of disparate storage resources managed by cooperating but independent administrative domains transparently accessible via a common namespace*. This approach to data access has become popular with high-energy physics experiments, particularly within the LHC. Federations provide a universal interface to data, regardless of data location; correspondingly, in the CMS [2] experiment, the infrastructure is known as "Any Data, Any Time, Anywhere" (AAA).

These data federations have been wildly successful with CMS users, liberating from the need to run jobs at a site to access data. In this paper, we explore the next step of the technology used. By integrating data federations with the CernVM File System (CVMFS) software, we provide stronger namespace semantics and a POSIX interface for these federations. CVMFS was modified in order to allow it to download files from a data federation; however, CVMFS is still used to store the namespace metadata of the filesystem, which is distributed through the CVMFS content distribution network (CDN). CVMFS also was extended to handle the user authorization necessary for data federations.

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

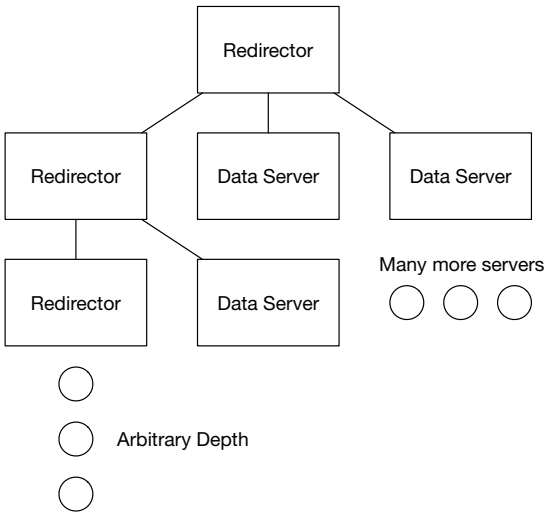


Figure 1: Diagram of XRootD's organization (Modeled after [4])

In this paper, we will discuss the improvements to user experience of data federations when combined with CVMFS. In Section 3 we will discus our CVMFS modifications. In Section 4, we will present the design of our experiments to show that using CVMFS to access data federations does not significantly decrease the transfer time of file. Section 5 will discuss the results from our experiments.

Finally, we will conclude in section 6 will discuss our ideas for future work with data federations and CVMFS. In section 7 we will list the contributions of the CVMFS modifications in improving the user experience of data federations.

2. Background

The technology that ATLAS and CMS experiments are using for their federation is XRootD [3, 1]. XRootD [4] is a scalable file server framework popular within HEP; here, the used daemons are `xrootd` for responding to user requests and `cmsd` used to cluster multiple `xrootd` services. The system is arranged into a tree hierarchy; `xrootd` leaf nodes export storage data. The nodes at the root and internal to the tree are known as "redirectors," and serve to redirect users to the appropriate child node. Figure 1 illustrates this hierarchy.

When a request for a file is issued from a client, it is first sent to the "local" redirector (often designated by the runtime environment). If the file location is not cached, the redirector broadcasts the request to connected data servers. If the file is located locally, the client is redirected to the appropriate data server. Otherwise, then the client is redirected to the regional redirector, which broadcasts to the connected sites (redirecting the client on a positive response). Clients continue up the tree until the global redirector is reached or the file is found.

XRootD efficiently allows users access to known files; while useful, it is a fairly limited functionality. It is impractical to get a list of files available in the federation: the xrootd protocol has a "list directory" API, but it would not be scalable to query each data server. The architecture of the XRootD federation limits how much of the namespace is visible at a time, since each the client may query a redirector at any level of the tree. Further, servers may join and leave the federation at any time, making a list of files obsolete.

As the namespace semantics of these federations are weak, most HEP experiments have a separate file catalog to manage the authoritative source of information. These centralized catalogs are not particularly scalable, nor are they as user friendly as navigating POSIX directory

structure.

The Cern Virtual Machine File System (CVMFS) [5] is heavily utilized in the community to distribute physics software. It is FUSE-based [6], providing a POSIX interface for local users. Within CVMFS, all writes occur at a central *repository server*; the internal data distribution is architected to allow efficient use of multiple layers of caching in order to minimize transfers. For example, it caches files locally on the worker node, site HTTP proxies, and finally a content distribution network (CDN) layer. All files are content-addressed, allowing infinite cache lifetimes if a file's content hash is known.

The CVMFSnamespace metadata is kept in a SQLite database [7], which is distributed through the same mechanism as the file data. The metadata includes a mapping of the filename to the content hash; this checksum is used as an integrity-checking mechanism in addition as a hash key for the cache hierarchy. Catalogs can also contain references to other catalogs, allowing the namespace to be partitioned over multiple files which only are downloaded on-demand. Periodically, the client checks for updates to the *root* catalog.

StashCache [8] is an OSG data federation designed for opportunistic organizations running on the OSG. It uses regional data caches to optimize delivery of data to clusters across the OSG. Figure 2 shows the organization of the StashCache federation; it is a twist on the AAA architecture used by CMS. The StashCache federation (and series of caches) helps provide scalability to source servers if the workflow's working set size is less than 10TB.

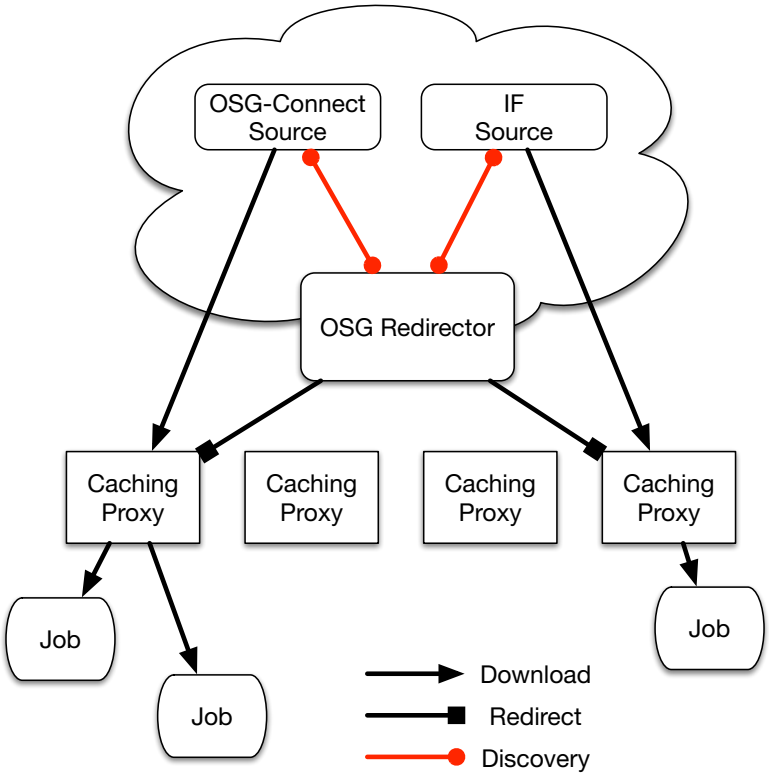


Figure 2: Diagram of StashCache's organization

Two sources are connected to the OSG Redirector, the OSG-Connect [9] source and the Intensity Frontier (IF) source. Requests for jobs are first directed to the nearest caching proxy. If the file access is a cache miss, the proxy contacts the OSG Redirector, which redirects the

proxy to the source which reports to have the requested data. The caching proxy downloads the data directly from the source while proxying the data to the job. Future requests for the same data will be served from the cache of the caching proxy rather than going to the source.

StashCache is frequently used by both OSG-Connect and the Intensity Frontier users. StashCache has many benefits for users and sites. For users, the caches are guaranteed to have fast networking, which can provide shorter download times. For sites, there are no requirement for running a service or host to use StashCache: if there is no cache, then data is simply streamed from the closest site. The regional caches are located at large sites, and administered jointly by the OSG and grid administrators at those sites. To simplify the user interface, special tool was developed for use with StashCache, `stashcp`. This tool implements the best practices of error handling for fetching data from the StashCache infrastructure.

3. Implementation

Several changes where required for CVMFS to work with a data federation. Three improvements deal with the ability to publish and serve files without copying them into the CVMFS repository server:

- Graft external filesystems onto CVMFS
- External server support
- Disable decompression for specially marked files

Finally, the CVMFS client process must be able to utilize credentials from the user's process environment.

If the federation contains multiple petabytes of data, copying all files to the repository server for checksumming is prohibitively expensive. We added a special "graft" file type which contains the metadata necessary for publishing but does not contain the file contents. To create a graft file, the `cvmfs_swissknife` command-line tool is passed the input file contents and where to store the resulting placeholder file. The placeholder file contains the size of the original file, as well as the checksum for each 32MB chunk of the file. The location of the placeholder file within the CVMFS repo is used for the remote URL.

Grafting allows publication of the file without copying it to the CVMFS CDN: clients must still know how to find it. When publishing, the graft file is marked as *external*. When a client encounters the file, it will download the file *based on its name, not content address* from a configured HTTP server outside the CVMFS CDN. If there are multiple endpoints (such as multiple StashCache caches), a GeoIP service is used to determine the ideal ordering of servers. Using GeoIP-sorted list of server endpoints works better in our environment than anycast approaches more common with the large CDN providers as the endpoints are run by an external entity.

The external access mechanism allows the CVMFS to access the files in-place in the data federation. However, CVMFS files are typically compressed at the repository and the CDN and only decompressed by the client: since the data federation does not keep a compressed and uncompressed copy, we added another piece of metadata to the catalog to indicate the files within the data federation are uncompressed.

Combined, these three features allow us to publish files in the data federation and access them transparently from a client. However, most physics experiments have strict authorization policies on their data federations that are otherwise not present in CVMFS's original use case (application software is typically considered public). We identified two distinct needs:

(i) **Private Data**: The contents of the files are private, but the names and directory structure is not. An unauthorized user may be able to see the directory structure, and the metadata associated with the file, but not the conents of the file.

(ii) **Private Namespace**: The contents and the namespace is private. Unauthorized users cannot see the directory structure, names, or the contents of the files.

In both cases, the authorization was coarse-grained: all members of the experiment were allowed access to all of the experiment's files. Hence, we only need to check for membership at the top-level.

CVMFS was modified to validate a user's credentials based on an examination of the accessing process's runtime environment. At the worker node, the user's identity is compared against an access control list stored as a POSIX extended attribute inside the CVMFS repository itself. If the user is allowed access and the requested file is not cached locally, the credential is again used to secure the HTTPS connection to the data federation. The credential is again validated by the data federation; the data federation does not distinguish between connections from CVMFS and a standalone client such as `curl`.

Changes are required to the CVMFS server to enable a private namespace as the namespace is otherwise kept in the HTTP-based CVMFS CDN. In this case, the client must be configured to authenticate with the repository server directly. There repository server will perform authorization; with the venerable Apache HTTP server, this can be done with the `mod_gridsite` [10] plugin.

The OSG has a data federation used by opportunistic organizations: StashCache [8]. The StashCache data federation uses XRootD servers for both the caching and data nodes. XRootD in the StashCache federation was configured to respond to HTTP requests. The XRootD configuration is augmented with the line: `xrd.protocol http:8000 libXrdHttp.so`. This line will enable XRootD to answer HTTP request for files.

4. Experimental Design

In our experiments, we demonstrate accessing data federations through CVMFS is efficient by comparing the download time of three data transfer methods:

(i) **CVMFS Access:** Test the speed of a a simple `cp` from CVMFS into the current working directory. This test uses the StashCache data federation via HTTP.

(ii) **Direct Federation Access:** Use the `stashcp` tool to download from StashCache using the xrootd protocol.

(iii) **HTTP Proxy:** Use `wget` to download the test file via HTTP proxies when available.

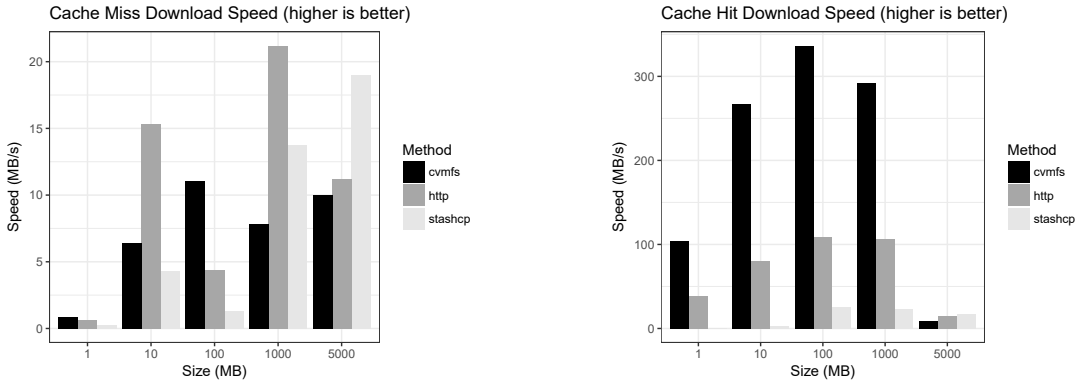
In all these transfer methods, the origin of the files are the OSG-Connect "Stash" data storage; in the tests, we vary the files of sizes from 1MB to 5GB. We submitted jobs to 4 sites across the OSG that are commonly used opportunistically: University of Nebraska, Clemson University, Syracuse University, and the California Institute of Technology.

For each size of files, we download the file three times. The first download simulates a cold cache. The second and third downloads simulate a hot cache.

Each of the download methods cache data at different levels. For the CVMFS access over the data federation, the data is cached on the worker node and at the regional caches of StashCache. HTTP proxies exist at each of the clusters used for the experiments, therefore the data is cached at each cluster. The direct federation access caches files at the regional caches.

5. Results

For the first results, we want to make sure that when the caching is working as expected. Download speeds at Clemson when the cache is populated vs unpopulated is shown in Figure 3. When there is a cache miss, the download speeds of the different methods are relatively close, as expected.



(a) Cache Miss (b) Cache Hit

Figure 3: Clemson Cache Hit and Miss Download Speeds

The low download speeds of the `stashcp` tool in Figure 3a when the file is small is due to the overhead of the tool itself. When starting, the `stashcp` tool has to query a web service to determine the closest cache. This overhead can be a large fraction of the time to download when the data size is small. `stashcp` is the slowest transfer method when the files are small. On the other hand, `stashcp` has a higher download speed than the other transfer methods when the file is large and no cache is available.

Figure 3b shows the download speed when the data has been cached. Since CVMFS caches the files on the node, it is very fast to download the files a second time, since it's just a local filesystem copy. HTTP caches the data at the cluster's HTTP proxy, therefore, the data needs to be transferred between the HTTP proxy and the worker node. The HTTP method is reliant on the speed of the network between the HTTP proxy and the worker node. `stashcp` needs to transfer the file over the wide are network (WAN) from the regional cache and is therefore dependent on the available bandwidth of the WAN.

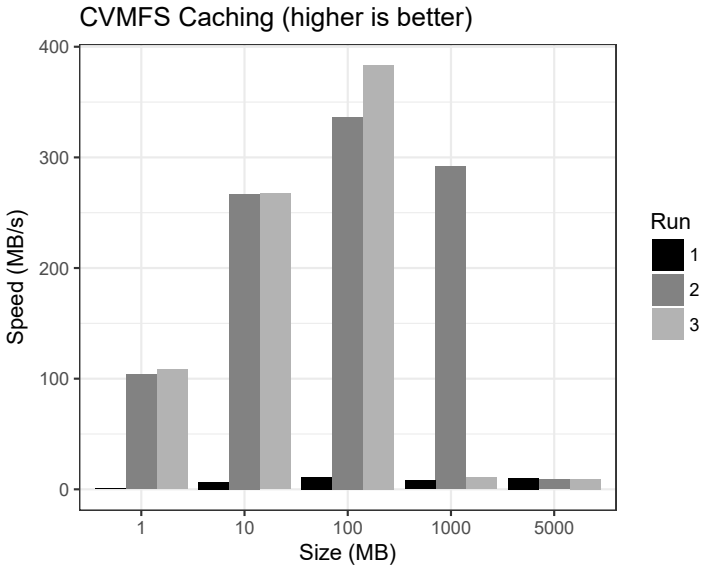


Figure 4: Clemson CVMFS caching

Since CVMFS caches data on the nodes directly, the data transfer can be faster than 300MB/s when copying files stored in CVMFS. For files up to 1000MB, caching in CVMFS works exceptionally well. But, as noted before, since CVMFS access to StashCache is meant for larger files, the local cache on CVMFS is set to 1GB. When downloading the 1000GB file, CVMFS caches it. But on the third download attempt, CVMFS evicts the cached file, and downloads it again since it is very close to the maximum cache size. For 5000MB, CVMFS is unable to cache the file at all, and it is streamed it to the client.

Some larger sites have a StashCache regional cache nearby. This significantly increases the transfer speed.

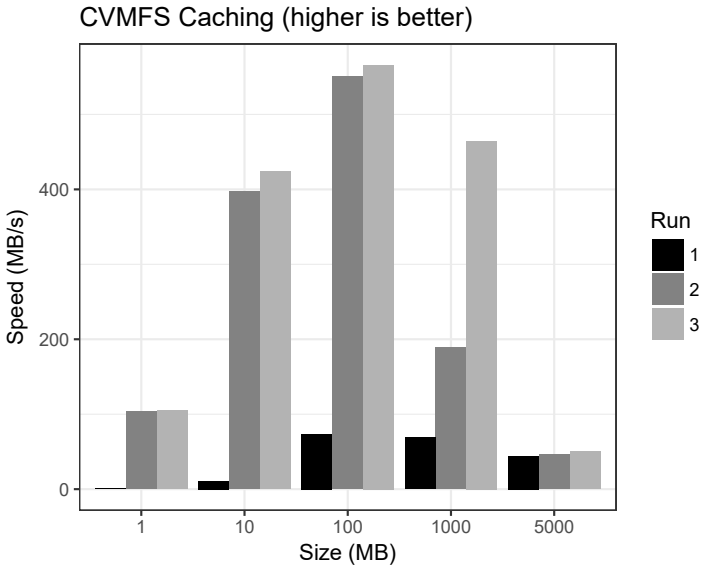


Figure 5: Nebraska CVMFS caching

For example, Figure 5 shows the CVMFS download speed at Nebraska, which hosts a StashCache regional cache. Since the cache is on-site, the download speeds are high even with cache misses. At Clemson, which does not have an on-site cache, the cache miss download speed was 7MB/s for the 1000MB file. At Nebraska, the cache miss download speed was 68MB/s. This difference can be attributed to slower WAN bandwidth between Clemson and the cache, which was at the University of Illinois Urbana-Champaign.

6. Future Work

The approach of using CVMFS for accessing data federations is in its infancy; we have demonstrated a viable technical approach, but questions of scale (for data volume and access rates) remain.

In the future, we hope to apply this to an entire physic experiment's dataset, utilizing the secure CVMFS features. We have demonstrated the ability to do an approximately 1PB CVMFS repository for the CMS experiment: unfortunately, this is a small subset of the approximately 40PB kept on tape by CMS: more work is needed to be able to generate the graft files in a distributed manner. Even not at the LHC data volume, we believe an experiment like LIGO would benefit from POSIX access for data kept in the StashCache data federation.

For cache-friendly workflows with terabyte-scale working set sizes, the elimination of the local worker node disk cache has become significant. Our setup puts significant load on the regional or site-level caches (site-level caches do not currently work with secure CVMFS *unless* the cache

registers with the VO). We are investigating adding a multi-tiered cache in the client. This would allow the CVMFS client to use the local site storage element as a large cache from the worker node. Assuming the site storage is well-matched to the compute resource's IO needs, we believe this will help the entire system continue to scale.

7. Conclusions

The experiments show that using CVMFS to access StashCache has the same performance as directly using the data federation when the when downloading data for the first time. After the local cache is populated, and as long as the data file is small enough to fit in the local cache, the performance of CVMFS outperforms both HTTP and federation access. When using CVMFS, the data files are cached at two levels, at the regional cache and at the local node, compared to federation access and HTTP which cache at one level each.

CVMFS caches data on the local node, repeated accesses to the same data will be cached. Since the number of cores on each computer is only increasing, repeated access could become more common. HTTP will cache data at the cluster level, therefore repeated access on any node on the cluster will utilize the cache. Therefore, for infrequent accessed data that fits in the HTTP cache, HTTP could have higher performance.

CVMFS access also provides an improved user experience by giving the user a posix-like interface to interact with. Since HTTP can provide better performance some times, and accessing the data federation directly can provide better performance in other cases, using CVMFS can be a tradeoff between user experience and performance. CVMFS provides a much improved user experience over both HTTP and federation access.

8. Acknowledgments

This work was supported by NSF award PHY-1148698, via subaward from University of Wisconsin-Madison. This research was done using resources provided by the Open Science Grid [11], which is supported by the National Science Foundation and the U.S. Department of Energy's Office of Science. This work was completed utilizing the Holland Computing Center of the University of Nebraska.

9. References

[1] Bauerdick L, Benjamin D, Bloom K, Bockelman B, Bradley D, Dasu S, Ernst M, Gardner R, Hanushevsky A, Ito H *et al.* 2012 Using xrootd to federate regional storage *Journal of Physics: Conference Series* vol 396 (IOP Publishing) p 042009

[2] Adolphi R *et al.* 2008 The cms experiment at the cern lhc, cms collaboration

[3] Gardner R, Campana S, Duckeck G, Elmsheuser J, Hanushevsky A, Hönig F G, Iven J, Legger F, Vukotic I, Yang W *et al.* 2014 Data federation strategies for atlas using xrootd *Journal of Physics: Conference Series* vol 513 (IOP Publishing) p 042049

[4] Dorigo A, Elmer P, Furano F and Hanushevsky A 2005 *WSEAS Transactions on Computers* **1**

[5] Aguado Sanchez C, Bloomer J, Buncic P, Franco L, Klemer S and Mato P 2008 Cvmfs-a file system for the cernvm virtual appliance *Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research* vol 1 p 52

[6] Szeredi M 2017 Fuse: Filesystem in userspace URL `https://github.com/libfuse/libfuse`

[7] Owens M and Allen G 2010 *SQLite* (Springer)

[8] Grid O S 2017 Installing stashcache URL `https://twiki.opensciencegrid.org/bin/view/Documentation/Release3/InstallStashCache`

[9] Jayatilaka B, Levshina T, Rynge M, Sehgal C and Slyz M 2015 The osg open facility: A sharing ecosystem *Journal of Physics: Conference Series* vol 664 (IOP Publishing) p 032016

[10] McNab A 2005 *Software: Practice and Experience* **35** 827–834

[11] Pordes R, Petravick D, Kramer B, Olson D, Livny M, Roy A, Avery P, Blackburn K, Wenaus T, Frank W *et al.* 2007 The open science grid *Journal of Physics: Conference Series* vol 78 (IOP Publishing) p 012057