A federated Xrootd cache

E Fajardo[1], A Tadel[1], M Tadel[1], B Steer[2], T Martin[1], F Würthwein[1]

[1] University of California San Diego, La Jolla, CA, USA
[2] Monash University, Wellington Rd, Clayton VIC 3800, Australia

E-mail: emfajard@ucsd.edu

Abstract. With the shift in the LHC experiments from the computing tiered model where data was prefetched and stored at the computing site towards a bring data on the fly, model came an opportunity. Since data is now distributed to computing jobs using XrootD federation of data, a clear opportunity for caching arose.

In this document, we present the experience of installing and using a Federated Xrootd Cache (A Xrootd Cache consistent of several independent nodes). There is some fine tuning towards and scaling tests performed to make it fit for the CMS Analysis case.

Finally, we show how this federated cache can be expanded into a federation of caches in which the caches can be distributed among computing centers.

1. Introduction and Physics motivation

The LHC physics program, especially the large general purpose collaborations ATLAS and CMS, depend on a distributed compute and storage infrastructure comprising roughly 150 Petabytes of disk space and 100-200,000 hyperthreads of Intel X86 compute power. For CMS, roughly 1/3 of this compute power, and nearly all of the disk space is deployed to support physics data analysis. Typically, a thousand scientists are active in a given month. More than 1000 peer reviewed science papers have been produced this way by the two collaborations.

In CMS, two data formats are used for data analysis, AOD and MINIAOD, that differ in size per event by x10. While MINIAOD is used by more than 90% of the analysis activity, AOD occupies most of the disk space. Most of the analysis activity is focused on a single coherent set of datasets produced with a consistent set of releases. CMS uses a data popularity mechanism to place replicas of datasets based on their popularity [1]. This mechanism works on timescales of one year or more before it deletes the datasets replicas from disk.

The present paper describes initial exploratory work towards a much more agile data replication mechanism based on caches inside the CMS global data federation. Given that the experiment knows in advance that the MINIAOD format will be heavily used in analysis. We set aside a relatively small fraction of the total disk space for caching a complete version of it with the objective of making that format maximally available at sites dynamically with minimal explicit maintenance beyond specifying the namespace cached. Hence the cache allows for a more efficient use of the sites disk space in which the data being cached is the one on demand right now as opposed to the data that was popular several months ago.

By allowing distributed caches across site boundaries among sites that are "close" in terms of network latency, we can further economize on the amount of disk space used for serving data analysis.

The present work describes the basic ideas, and initial benchmarking towards such a system.

2. Xrootd and Data Federations

As demonstrated by the Anydata, Anytime, Anywhere project (AAA) [2] XRootd [3] was the missing piece of the puzzle allowing the HEP community to step up from the concept of distributed, multi-tiered storage to the level of global data federations with a single data-access entry point and a common data-access protocol. Hierarchical deployment of XRootd redirectors allows for a real-time discovery of sites where required data is available now, bypassing entirely the file catalogues where information might be stale or the site listed there might be temporarily inaccessible. While XRootd supported multi-storage deployments before, the novel idea of AAA was to extend and improve this XRootd functionality to seamlessly function in a global, multi-site environment. On the storage side, XRootd frontends or data-access plugins had to be developed for all storage solutions used by CMS (hdfs, dCache, and DPM).

With AAA extending from the United States based pilot to a global CMS data federation in 2014 [4], the usage of AAA quickly grew from the initial use-cases (fallback to remote data access, overflow of jobs from oversubscribed sites[5], direct remote access for data exploration and visualization) to the point of profoundly affecting the Computing model of CMS and also using remote data access for production jobs as well as for data access from jobs running on opportunistic resources. With all relevant CMS data being accessible through AAA, the data placement and data movement strategies started to change as well, becoming based on data popularity [1] or actual data access (by using XRootd caching proxy – just in time data placement).

3. XRootd File-based Caching Proxy

XRootd file-based caching proxy (known as Proxy File Cache (PFC) within XRootd code and documentation) acts as an intermediary between local XRootd clients and a remote XRootd service, either a single data-server or a complete data federation. As a proxy cache it serves data available on local disk directly and forwards requests for missing data to the remote service, serving the data to the requesting client and writing it into the cache. Caching proxy will only open a remote connection if data is actually not available locally or when a segment of a file not already in the cache is accessed by a client. This guarantees that data in the cache remains usable even when the data source is not available. Data can be automatically prefetched when it is known that most of the file will be read once it is open. As all caches, the available space is automatically managed and least recently used files get purged once disk usage reaches a predetermined level.

A set of caching proxy servers can be connected into a Caching Proxy Cluster by using a standard XRootd manager as an entry point into the distributed cache. Further, using the same mechanism, several caching clusters can be connected into a federation of proxy clusters. This makes sense when caching clusters in the federation are significantly closer to each other than to the source data federation.

3.1. Local Xrootd Cache hardware setup

For the pilot XrootD cache system at UCSD we tried to maximize the number of spindles while minimizing the amount of XrootD redirections a job needs to read its input data. Older worker nodes were chosen that had the capacity for 12 3.5" disks drives of modest size. The systems use two Intel Xeon X5650(6C/12HT) processors, 48GB of RAM, and 12 2TB HGST Hard drives each. We have 11 systems in our Xcache for a total disk capacity of 264TB. In order to make best use of the disks we retrofitted the systems with Mellanox MT27520 10Gbps network interfaces and connected them directly to our Arista 7150S switch using 10Gbps Mellanox AOC Cables.

3.2. Local tricks to increase use of the cache

The main use case for the cache is for CMS. We wanted CMS jobs to run at the UCSD Tier-2 site (and use the cache) for jobs that run over the datasets that were in the cache but not necessarily advertised as such via the global CMS PhEDEx data management system [6] (i.e jobs that would not originally be routed to the UCSD Tier-2 cluster). We decided to serve all MINIAOD(SIM) data from CMS for the 2016 data taking period, and corresponding simulations. This namespace was configured to be cached in the XRootd cache. Configuration changes were then made to the CMS overflow mechanism [5] to steer jobs to UCSD that pend for more than 6 hours in the global HTCondor pool of CMS[7], and declare files in the cached namespace as input. In addition there were some modifications needed for the site local configuration, so jobs requiring these datasets would not try to access the local storage system but the Xrootd cache cluster.

4. Scale tests and estimated throughput

Given the potential for the cache (two sites, and hundreds of terabytes of data) ensuring that it would scale with the current and future availability of compute nodes was a must. In this section we present the methodology and the scale results obtained while stressing the proxy cache cluster infrastructure. The base case for the test was the fact that CMS jobs read at $1MByte/s$ per core (client) [2]. All the measurements were done using clients that read at least twice this rate.

The scale tests for this project were divided into two different sections. Section involved holding the number of concurrent data requests fixed, while varying the rate at which individual processes requested data. Section4.2 investigated holding the total requested data rate fixed while varying the number of concurrent requests and individual data request rates.

Multiple large file requests were made from host machines at UCSD prior to the experiments to ensure that the cache was hot (desired files already present in the cache) when experiments were performed. This guaranteed a high cache hit/rate since requests were made for files from this same dataset.

GlideinWMS[8] was used to submit several jobs (clients) at the same time to request files from the cache, i.e create the artificial load. To gather the highest amount of distributed cores while reducing the impact on production operations in the cluster a sleeper pool was used. A sleeper pool consist of a batch system pool in which compute nodes are configured to advertise more cores than they have, under the premise that those extra slots will consume a negligable amount of resources(CPU, memory and disk). The sleeper pool is thus "overlayed" on top of the UCSD Tier-2 production system.

4.1. Fixed number of concurrent slots

For the this part, the number of concurrent jobs is fixed to 1000 and the data request rate is varied. The rate being specified in units of MB/10s reflects the nature of the script submitted in each job, which sleeps for 10 seconds between making remote file requests. The expected data-rate per job (see green data in Figure 1) is simply the specified data request rate divided by 10, changing to units MB/s.

4.2. Fixed Total Data Rate

In this section the total data request-rate is fixed at $5GB/s$ and the other parameters are varied. For example, this can be achieved with 1000 concurrent jobs making average requests of $5MB/s$, or equally with 5000 concurrent jobs making average requests of $1MB/s$. We would expect the system throughput to decrease when the number of clients increases because of the degradation of the service when disks in the cache cannot handle enough read requests. This is exactly what we see in Figure 2. Notice that in the worst case scenario at $1MB/sec$ (the expected data

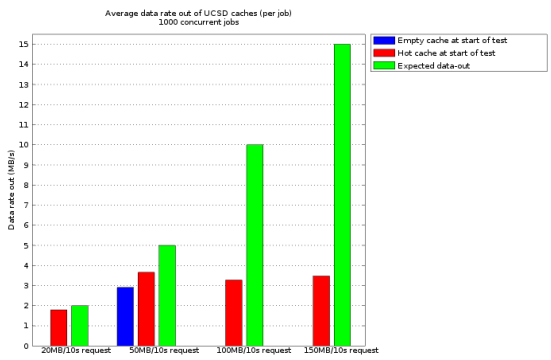

Figure 1. Average data-rate throughput (per job) for 1000 jobs

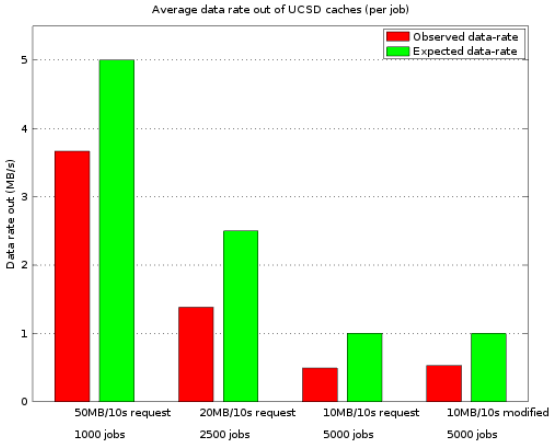

Figure 2. Average data-rate (per job) for a fixed total data request-rate.

rate) and usage of the entire cluster (5000 clients) the cache can only deliver up to 60% of the expected data-rate.

5. Conclusions and Future Work

We see in the future more sites setting up similar cache clusters like this one. In particular Caltech setting one up in which jobs running at either site can access seamlessly files from each other cache. This will require doing some these tests again across the Wide Area Network.

More work and tests will be needed to understand and mitigate the performance degradation when running at extreme concurrency levels if the data rates per core of CMS were to greatly increase in the future. However, given the rates seen today on the production Tier-2s we are confident based on these results that the existing cache will operate smoothly and efficiently. We are thus since running this cache as part of the regular production Tier-2 system for CMS.

Acknowledgments

This work is supported in part by the National Science Foundation through awards PHY-1148698 and PHY-1624356.

References

[1] Kuznetsov V, Li T, Giommi L, Bonacorsi D and Wildish T 2016 *Journal of Physics: Conference Series* **762** 012048 URL `http://stacks.iop.org/1742-6596/762/i=1/a=012048`

[2] Bauerdick L, Benjamin D, Bloom K, Bockelman B, Bradley D, Dasu S, Ernst M, Gardner R, Hanushevsky A, Ito H, Lesny D, McGuigan P, McKee S, Rind O, Severini H, Sfiligoi I, Tadel M, Vukotic I, Williams S, Wrthwein F, Yagil A and Yang W 2012 *Journal of Physics: Conference Series* **396** 042009 URL `http://stacks.iop.org/1742-6596/396/i=4/a=042009`

[3] Dorigo A, Elmer P, Furano F and Hanushevsky A 2005 Xrootd/txnetfile: A highly scalable architecture for data access in the root environment *Proceedings of the 4th WSEAS International Conference on Telecommunications and Informatics* TELE-INFO'05 (Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS)) pp 46:1–46:6 ISBN 960-8457-11-4 URL `http://dl.acm.org/citation.cfm?id=1391157.1391203`

[4] Bloom K and the Cms Collaboration 2014 *Journal of Physics: Conference Series* **513** 042005 URL `http://stacks.iop.org/1742-6596/513/i=4/a=042005`

[5] Sfiligoi I, Wuerthwein F, Bockelman B, Bradley D C, Tadel M, Bloom K, Letts J and Tadel A M 2012 *Journal of Physics: Conference Series* **396** 032102 URL `http://stacks.iop.org/1742-6596/396/i=3/a=032102`

[6] Rehn J 2006 Phedex high-throughput data transfer management system

[7] Balcas J, Belforte S, Bockelman B, Colling D, Gutsche O, Hufnagel D, Khan F, Larson K, Letts J, Mascheroni M, Mason D, McCrea A, Piperov S, Saiz-Santos M, Sfiligoi I, Tanasijczuk A and Wissing C 2015 *Journal of Physics: Conference Series* **664** 062031 URL `http://stacks.iop.org/1742-6596/664/i=6/a=062031`

[8] Sfiligoi I, Bradley D C, Holzman B, Mhashilkar P, Padhi S and Wurthwein F 2009 The pilot way to grid resources using glideinwms *2009 WRI World Congress on Computer Science and Information Engineering* vol 2 pp 428–432