# Xcache in ATLAS Distributed Computing

*Andrew* Hanushevsky[1], *Hironori* Ito[2], *Mario* Lassnig[3], *Radu* Popescu[3], *Asoka* De Silva[4], *Michal* Simon[3], *Robert* Gardner[5], *Vincent* Garonne[6], *John* De Stefano[2], *Ilija* Vukotic[5], *Wei* Yang[1*] on behalf of the ATLAS Collaboration

[1]SLAC National Accelerator Laboratory, Menlo Park, CA, USA
[2]Brookhaven National Laboratory, Upton, NY, USA
[3]CERN, Geneva, Switzerland
[4]TRIUMF, Vancouver, BC, Canada
[5]University of Chicago, Chicago, IL, USA
[6]University of Oslo, Oslo, Norway

**Abstract.** Inherited from the flexible architecture of Xrootd, Xcache allows a wide range of customization through configurations and plugins modules. This paper describes several completed and ongoing R&D efforts of using Xcache in the LHC ATLAS distributed computing environment, in particular, using Xcache with the ATLAS data management system RUCIO for easy-to-use and to improve cache hit rate, to replace Squid and improve distribution of large files in CVMFS, to adapt the HPC environment and the data lake model for efficient data distribute and data access for the HPCs.

## 1 Introduction

Xcache is a Squid-like cache, but it primarily uses the "xroot" (a.k.a. "root") protocol [1], HTTP protocol being added on. It is a multi-threaded file caching application that can asynchronously fetch and cache file segments or whole files. Its primary designed use cases are caching static scientific data file of any format, large or small. Xcache is built upon Xrootd [2] and is flexible to be customized for many usage scenarios, via configuration or plugins. A single Xcache can easily be deployed via container or CVMFS [3] for a user or a small user group, while a cluster of Xcache can be built for large or heavy use cases.

This paper describes some of the Xcache related efforts related to the distributed computing of the LHC ATLAS experiment [4]. For example, Xcache works with RUCIO [5] to improve cache hit rate and provide a location independent data access via the global logical file name. Xcache can also use HTTP protocol with clients, and this capability is explored to replace Squid cache in the CVMFS data distribution chain. HPC environments are usually different from the Grid sites [6]. This paper will discuss work being done, as

* Corresponding author, yangw@slac.sanford.edu

well as plans on using Xcache with HPCs and the data lake model [7] to efficiently distribute data and access data on HPCs.

## 2 Xcache utilizing information from RUCIO

RUCIO is a data management system developed by the ATLAS experiment. In the ATLAS distributed environment, multiple copies of a data file are distributed to several locations. Upon a query, RUCIO can provide a list of all locations filtered by protocols (root, HTTP, gsiftp, etc.). The output list is in metalink XML format [8]. The list can be sorted based on the distance to the requestor, using GeoIP [9]. A plugin of Xcache has been developed using the global Logical File Name (gLFN) in the form of /atlas/rucio/scope:file. The plugin queries RUCIO to find the optimal data source and other data sources. The gLFN is a concept initially developed by the ATLAS FAX project [10], and represents a location independent file path for users to access the data file via the FAX system and its redirection network ("/atlas/rucio" identifies that this is a RUCIO managed file for the ATLAS experiment; "scope:file" is the RUCIO Data Identifier (DID) [11], which identifies the file in RUCIO system). The plugin is capable of failover to the next available data source should the previous one not respond. If a user has a specific data source in mind, they can prefix the Xcache URL with the actual data source URL. In that case, the plugin will not query RUCIO but will go directly to the user specified data source.

In storage systems managed by RUCIO, data files are stored in a predicable location based on their RUCIO DID. Using this pattern, the plugin can identify the same files distributed at different remote data sources and use the same cache entry for them, as shown in Figure 1. Files accessed using gLFNs also follow the same pattern and share cache entries in the same way.

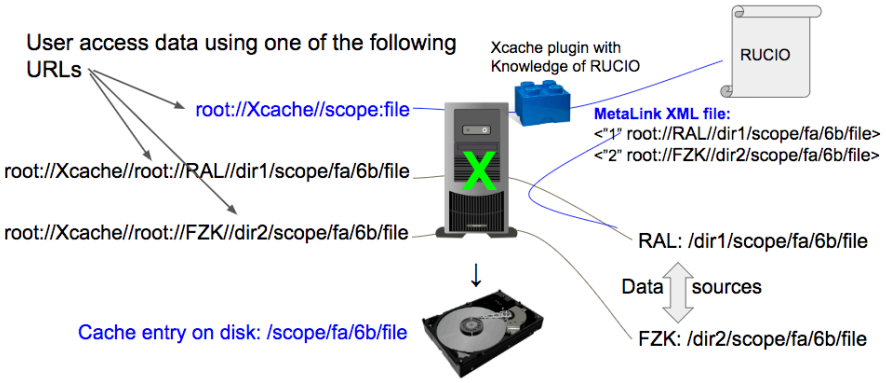


Figure 1. A Xcache plugin that communicates with RUCIO to retrieve file location XML (metalink). It also understands how RUCIO store data files. This knowledge enables Xcache to share cache entries for the same RUCIO Data IDentifier (DID). Note that the path "scope/fa/6b/file" is a predictable path because "fa6b" are the first four hexadecimal numbers of the MD5 checksum of string "scope:file".

As a part of the integration with ATLAS workflow, we also utilize the concept of volatile storage in RUCIO and report the change in cache space to RUCIO. This makes RUICO aware of the content in the cache, and can potentially help ATLAS workflow management system Panda [12] to schedule jobs accordingly.

## 3 Xcache with HTTP data source and CVMFS

The Xrootd / Xcache software has a plugin module XrdHTTP that enables it to communicate with client via the HTTP protocol. Using this plugin, we were able to export the entire CVMFS Stratum-0 via the Xroot protocol, and then use a XrdHTTP enabled Xcache to serve the data to the CVMFS client on batch nodes. The green arrow in Figure 2 shows the schema.

We conduced such a test that exported the ATLAS CVMFS Stratum-1 (a replica of Stratum-0) at the Brookhaven National Laboratory via a read-only Xrootd server, using the root/xroot protocol. We then ran a XrdHTTP enabled Xcache at the SLAC National Accelerator Laboratory, and modified the configuration of the CVMFS clients on a cluster of SLAC's batch nodes to use this cache instance. This batch cluster at SLAC was put in production operation for several weeks. The CVMFS clients on this cluster function exactly the same as before. This demonstrated that we can use XrdHTTP enabled Xcache to replace the various Squid cache servers in the CVMFS data distribution chain. It is also possible to replace the functionality of Stratum-1 if needed. Giving the Xcache handles large files much better than the Squid cache, this test opened the possibility of using CVMFS to distribute large data files.

The manifest file (.cvmfspublished) is the entry point to a CVMFS repository. When CVMFS repository is updated, a new manifest file is put in the root directory of the CVMFS repository. Because Xcache is designed for static files, we periodically delete this manifest file from the above Xcache. This will trigger a refresh of the manifest file, and thus make the CVMFS client aware of the changes made to the CVMFS repository. Old files in the Xcache which are no longer part of the CVMFS repository will eventually be purged by the Xcache when space is needed.

As a part of the work of using Xcache with CVMFS, a prototype of HTTP plugin for the Xrootd client (as opposed to the XrdHTTP, which is a plugin to the Xrootd server) was developed in order to have Xcache work with the HTTP data source, and be able to seamlessly work with the existing Apache infrastructure for CVMFS. The red arrow in Figure 2 shows the data flow.

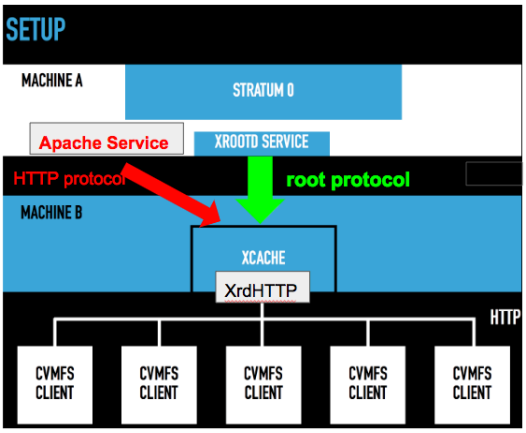


Figure 2. By running a Xrootd service on CMVFS Stratum-0/1, we can use a XrdHTTP enabled Xcache to replace Squid cache in the CVMFS distribution chain. When the HTTP plugin for the Xrootd client will be fully developped, it will be also possible to have the Xcache seamlessly integrated into the CVMFS Apache infrastructure.

## 4 Xcache in HPC environment

Most of the Xrootd and Xcache development has been around the traditional Grid site setup, where there is a standalone machine or a cluster of standalone machines to function as Xrootd servers or Xcache. In this setup, each machine uses a locally attached disk as the cache storage. This cache storage is only accessible by the machine itself, and a dedicated filesystem is created on this storage for caching purpose.

In a HPC environment, we usually don't have such a dedicated setup. Instead, the nodes that we are allowed to run Xcache service are usually edge nodes (such as login nodes and data transfer nodes) with Internet connections. The storage available to users usually resides at several very large shared Lustre or GPFS storages that are connected to the batch nodes and edge nodes via InfiniBand-like low latency network. TCP/IP network among the nodes may or may not be available. If the TPC/IP network is available, it is usually not the optimized interconnection infrastructure among the nodes. We identified several challenges and optimization steps in order for Xcache to efficiently use the HPC resources.

1. *Cache space accounting challenge*: we can no longer depend on the space usage of a dedicated file system for the cache space accounting. A new mechanism is needed to trace the cache space usage.
2. *Enable clients to directly access the cache data via the shared Posix file system*: For those data already in the cache, the data doesn't need to go through the Xcache nodes if the client has direct access to the shared file systems. A new function in the Xrootd client and server was developed so that the Xcache will redirect the clients to the actually location in the shared file system if the data file is fully cached. This approach not only optimizes the data flow path, but also enable the client to use the most efficient, native data access mechanism provided by the HPC sites.
3. *Enable clients to fetch data from Xcache via low latency network*: new development work is needed to enable Xroot protocol over the low latency network via the Remote Direct Memory Access (RDMA). With this function, the Xcache can deliver newly fetched data blocks to the clients before it has a chance to save it to the shared file systems. Delivery of partially cached files can also benefit from this mechanism. This will avoid forcing the clients to understand how Xcache keeps trace of partially cached files.

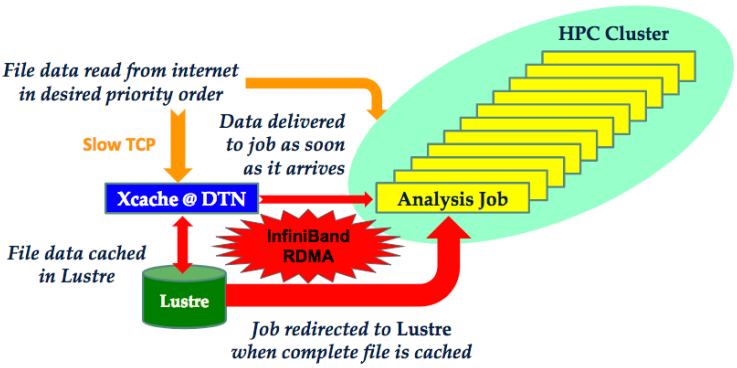


Figure 3. In a typical HPC setup, Xcache runs on edge nodes such as data transfer nodes (DTNs) and uses shared file systems (such as Lustre above) as cache space. For efficient data access, it is desired that the clients (e.g. Analysis jobs above) read fully cached data files directly from the Lustre file system, and read newly fetched data blocked from Xcache via the low latency network, such as InfiniBand.

A typical HPC environment with Xcache and possible optimization is shown in Figure 3. A cluster of Xcache is currently running on the data transfer nodes at the National Energy Research Scientific Computing Center (NERSC) in the US. Each Xcache node uses a dedicated directory in the shared Lustre file system as cache space. All data traffic is currently going through the TCP network among the nodes. This setup is used as the development and testing environment for the challenges and optimization we identified above.

## 5 Summary

We identified several areas where Xcache can be used by ATLAS and other scientific experiments. The RUCIO plugin for Xcache targets individual users or user groups so that they can just access their data without having to find out the ATLAS data placement and follow the change of the data placement, and without having to manage the storage space. Xcache for CVMFS opened the possibility of using CVMFS to distribute large data files, and the possibility of a centralized local CVMFS cache (as opposed to cache space on individual batch nodes). Xcache for HPC aims at optimizing the Xcache for the HPC environment by rerouting the client to directly read from the HPC shared file system when possible, and by utilizing the low latency network to deliver data.

Along with this work, we are advancing the concept of volatile storage, a useful concept when exploring opportunistic resources, expanding Xcache to work with data sources that supports HTTP protocols, and developing Xroot protocol over RDMA.

## 6 Acknowledgement

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

## References

1. Xrootd.org http://www.xrootd.org
2. L. Bauerdick, K. Bloom, B. Bockelman, D. Bradley, S. Dasu, J. Dost, I. Sfiligoi, A. Tadel, M. Tadel, F. Wuerthwein, A. Yagil, Journal of Physics: Conference Series, Volume 513, Track 4
3. S. Aguado, J. Bloomer, P. Buncic, L. Franco, S. Klemer, P. Mato, Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research vol 1 p 52
4. The ATLAS Collaboration, 2008 *JINST* **3** S08003
5. V. Garonne, R. Vigne, G. Stewart, M. Barisits, T. Beermann, M. Lassnig, C. Serfon, L. Goossens, A. Nairz (on behalf of the Atlas Collaboration). *Journal of Physics: Conference Series* **513**4
6. Worldwide LHC Computing Grid (WLCG), http://wlcg.web.cern.ch
7. M. Girone, S. Campana, Data Management plenary and parallel sessions, https://indico.cern.ch/event/658060/contributions/2940554/attachments/1625111/2587608/WLCGDataManagementSummary.pdf
8. The Metalink Download Description Format, https://tools.ietf.org/html/rfc5854
9. What is GeoIP? https://docs.nexcess.net/article/what-is-geoip.html

10. R. Gardner, S. Campana, G. Duckeck, J. Elmsheuser, A. Hanushevsky, F. Hönig, J. Iven, F. Legger, I. Vukotic, W. Yang, Journal of Physics: Conference Series vol 513 (IOP Publishing) p 042049
11. RUCIO Document, https://media.readthedocs.org/pdf/rucio/latest/rucio.pdf
12. A. Klimentov, A. Vaniachine, K. De, T. Wenaus, S. Panitkin, D. Yu, G. Záruba, M. Titov, SC Companion (IEEE Computer Society) 1521-1522