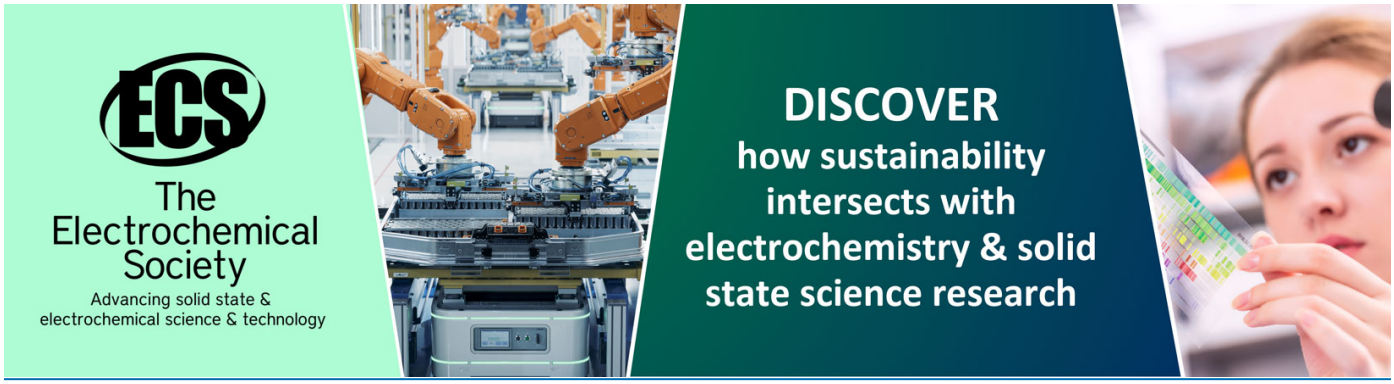
Caching Servers for ATLAS



IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 062017 doi:10.1088/1742-6596/898/6/062017

Caching Servers for ATLAS

R W Gardner[1], A Hanushevsky[2], I Vukotic[1], and W Yang[2]

[1]University of Chicago, 5735 S Ellis Ave. Chicago IL 60637, USA
[2]SLAC National Accelerator Laboratory, 2575 Sand Hill Road, Menlo Park, CA 94025, USA

rwg@uchicago.edu

Abstract. As many LHC Tier-3 and some Tier-2 centers look toward streamlining operations, they are considering autonomously managed storage elements as part of the solution. These storage elements are essentially file caching servers. They can operate as whole file or data block level caches. Several implementations exist. In this paper we explore using XRootD caching servers that can operate in either mode. They can also operate autonomously (i.e. demand driven), be centrally managed (i.e. a Rucio managed cache), or operate in both modes. We explore the pros and cons of various configurations as well as practical requirements for caching to be effective. While we focus on XRootD caches, the analysis should apply to other kinds of caches as well.

1. Motivation

Beyond static placement of data on grid based storage elements, dynamic data delivery to distributed processing resources for all types of ATLAS [1] event production and analysis remains a significant challenge. In the standard LHC grid computing model the input data file and processor are co-located within the same grid site. Files are preplaced by rules, or "subscriptions", managed by a central file catalog and replication service, such as Rucio [2] developed for ATLAS. The obvious restrictions are that tasks requiring those input data must compete with other (potentially higher priority) workloads scheduled to the site and therefore may experience long wait times in queue. In the ATLAS processing system PanDA [3], this can be mitigated by re-scheduling a task to a "less busy" site and accessing the input data over the wide area network, provided the cost/benefit is sensible. Indeed, we have created a redirection network [4] based on XRootD [5] that federates over 90% of the disk resident storage in ATLAS which opens the network as a third resource to be exploited by the collaboration. The benefits to such an approach are clear: sites within a nearby region with sufficient network bandwidth capacity can aggregate storage resources, reducing storage costs while more efficiently handling centrally and locally scheduled workload tasks. The question we seek to answer in this paper is whether wide are data access to remote storage resources can be made more efficient with local caching.

Other motivations which have led us to this path of development are:

- Many small computing sites have limited storage capacity and spend a great deal of time managing datasets downloaded from the grid, keeping locally stored files in sync with the central catalog, and managing deletions and subscription rules. Could these sites reduce the number of local custodial data sets, and thus the local data management burden using a

relatively simple, stateless cache while relying on larger centers in the region to host complete datasets?

- Most analysis datasets are read multiple times, as are pileup datasets and data that might be used for event mixing. A local caching service storing only data of interest would typically be relatively slow on the first pass (if the data are remote) while subsequent reads, being on local disk or on network attached storage within the local area network, would become more efficient.
- In ATLAS, a typical analysis job may consume only 10% of the file content. A caching service that selects and stores only those data blocks consumed by the job therefore can reduce the storage capacity requirement of the local center for a given analysis.
- In the past two years the capacity of the ATLAS Tier-1 and Tier-2 centers has increased significantly. These centers host the most popular datasets used for analysis and they can be kept on disk for periods of time commensurate with a full analysis to publication cycle.
- As Rucio stores the physical locations of every file replica in ATLAS, the grid based analysis and group production tasks (so called "derivations", i.e., the usual slimming, skimming, and thinning of reconstructed data) can directly read from remote storage.
- All of the data in U.S. ATLAS, and indeed at the majority of sites used by ATLAS, are directly accessible via the XRootD protocol, making these data available to off-grid end-users at Tier-3 centers or from other job queues such as HTCondor.
- The caching proxy server can fetch missing chunks of data, fetching the missing pieces from the federation if or when needed.
- ATLAS often makes use of high performance computing resources where there is no dedicated storage hardware or operational manpower. In these centers local proxy caches can be a good substitute for a fully integrated Rucio storage element, with its attendant operation and data management tasks.

Finally, as data ingress costs to commercial cloud providers are free, cloud based storage is not and costs can add up quickly if not managed and monitored carefully. By caching only data needed, proxy caches could use cloud-based storage more economically.

2. XRootD Caching Proxy Server

2.1. Cache Setup and Redirection Network

In Figure 1 we show the physical caching arrangement in the context of an ATLAS Tier-2 center, an XRootD data cache [6], and an XRootD-based redirection network. Analysis jobs (clients) are configured with input file lists, each file expressed as a global logical filename (gLFN) and the address of the local XRootD proxy cache. In ATLAS, gLFNs allow regional and global filename discovery through the federation network. Participating storage endpoints in the federation have site-level redirectors which deterministically translate the gLFN to a local physical path specific to that site. Thus in Figure 1, clients first attempt

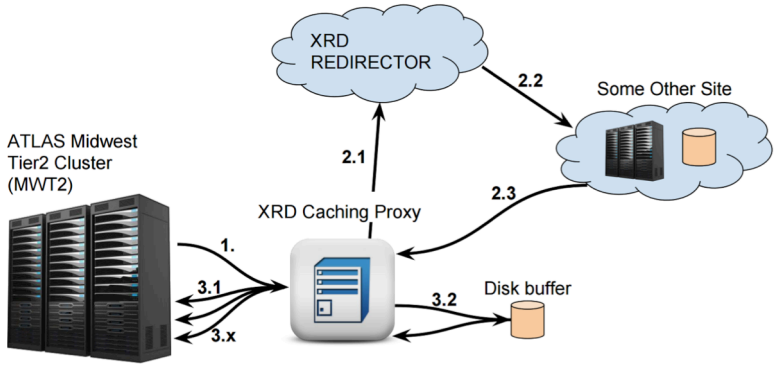


Figure 1. Setup of a caching proxy server in a Tier-2 context with redirection network.

to open directly (or, optionally copy) a file by contacting the local proxy server (**1**). For fresh data sets (no local copy, i.e. a "cold" cache) the proxy server will redirect to the federation. The proxy server is itself configured to point to a redirector nearby (**2.1**), in the Tier-2's region. This redirector advertises the request to the federation (**2.2**) which returns the XRootD endpoint address of another site having the file to the proxy server (**2.3**). Data from the file is delivered to the client (**3.1**) and simultaneously given to a write queue to the proxy server disk (**3.2**).

2.2. Standard operations

File open: In the cold cache scenario just described a WAN (wide area network) access is required to open the file. If the cache is warm (non-empty), the server will open the file on local disk if present. Note that the remote file open is only initiated if the requested block is not available in the cache.

Sequential and vector reading: The proxy cache uses fixed size buffers which can be configured between 32 kB and 4 MB. If the block is in the proxy server's RAM or on its local disk, it will serve from those locations. Otherwise, the data is requested from the federation and served directly to the client and to an in-memory write queue, there remaining in RAM until flushed to disk. All requests are padded to align with cache blocks. In the warm cache case, subsequent accesses for the same data are accessed locally. The caching server has read ahead buffering which reduces latency.

2.3. Architectural features of the server

The caching proxy server is based on the standard XRootD proxy server, but with a special **caching plugin** developed by the UCSD group as part of the CMS AAA project. Figure 2 offers a detailed view of the stacked architecture of the service component layers. The caching plugin interacts with the local disk to write blocks or whole files of requested data. Client requests can use either HTTP, HTTPS or XRootD protocols (in this article we used the XRootD protocol) and are managed by the XRootD protocol framework driver. The XRootD proxy server has the concept of a logical filesystem with a plugin-type architecture. The caching plugin uses a POSIX client API for reading from remote or local disk.

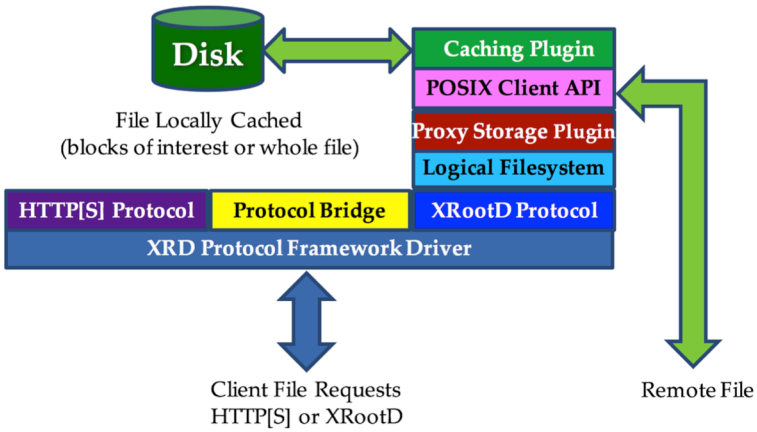


Figure 2. The caching server architecture with the major components.

The major functional actions of the proxy server are summarized:

1. The caching proxy does not contact the remote source until there is a cache miss. So, if all the requests can be satisfied from the local disk or memory no remote servers are accessed.
2. The caching proxy tries to anticipate future references. This is done by prefetching which is configurable. Blocks that get pre-fetched are given priority for in-memory access by the client and written to disk in the background. In some ways this works primarily as a memory cache with lower priority given to long-term caching (i.e. disk persistence is not as important as providing the data to the client when it is needed).
3. The cache consists of two parts: a) the actual data which is written in a directory corresponding to the file path, and b) the meta information which describes what is actually

present. The meta information is written into a configurable separate directory as *cinfo* files. The split between data and meta data allows one to speed up access by placing the metadata in high speed storage (e.g. a solid state storage device) while the actual data can go into slower speed storage (e.g. spinning disk arrays). Obviously the metadata is a small fraction of the actual data so this is quite workable.

4. The caching proxy uses the standard XRootD storage plugin for all accesses to the underlying cache. This means one can configure the proxy to use whatever storage system is available (e.g. directly attached disk arrays, network attached storage, parallel file systems, object storage, etc.). One is not tied to a standard universal filesystem specification so optimizations are unlimited. While such optimizations have not yet been tried the fundamentals are present in the protocol.
5. The I/O is fully asynchronous. Therefore, reads, pre-fetching, and writing can all occur simultaneously. This can provide substantial latency reduction.
6. The caching proxy can be deployed as a cluster using the standard XRootD clustering technology. This gives the caching proxy scale-out capability. One can easily grow it to whatever size is needed.
7. Future enhancements include automatic "load shedding" either using pass-through or client redirection, integrated monitoring, additional plugins to deal with metadata files and additional performance enhancements.

Given that the proxy cache is based on the POSIX client API, it makes it possible to configure a laptop caching proxy (similar to AFS). That is, one doesn't need a server to provide caching functionality. One can have a personal caching proxy (a local cache). This has not been explored but is an interesting concept.

2.4. Configuration options

We describe here some of the configuration options available to site administrators choosing to deploy a proxy caching server for their cluster.

Authentication. For local accesses any method supported by XRootD can be used. For remote accesses, this is determined by the federation. If grid service authentication is used, then the proxy must have a valid service certificate registered in a virtual organization server.

Cache content: as already mentioned, the *cinfo* file stores details about blocks already downloaded, and all local access. The physical path to the cache location on disk is configurable. For each file, the *cinfo* file contains the buffer size used, number of buffers downloaded, completion status, and access statistics such as the number of times accessed, time-date of last access, number of bytes on disk, number of bytes held in RAM, and the number of bytes missed.

Prefetching: the proxy can optionally issue advance read requests to reduce latencies caused by network round trip times (RTT). The number of blocks to pre-fetch is a parameter.

Decision to cache plugin: a configuration option to select which parts of the global namespace to be cached, based on the path. By default, all files are cached.

Cache purging behavior: there is a high/low water mark algorithm to trigger or cease purging of the cache. The low and high water marks are configurable.

Cache write queue behavior: The *pfc.ram* directive exposes options to control write queue behavior.

Cache block size: all read requests are rounded to the block size (the default is 1 MB). The resulting performance will depend on the workload served by the cache. Smaller block sizes will burden the disks with many small writes. Larger block sizes will increase memory usage in the cache server, and will grab more data off the network when only parts of the file are needed.

2.5. Internal features of the caching plugin

The internal features of the caching plugin itself highlight the operational behaviour of the caching proxy server.

Cache memory management: A majority of the caching proxy RAM usage is for the write queue. Pre-fetched buffers are placed at the beginning of the write queue, as it assumed they'll be needed at a later time. This ensures the memory is vacated as soon as possible. Buffers obtained to service outstanding read requests are placed at the end of the write queue, as it is assumed they will be needed to serve future read requests and therefore should remain in memory for as long as possible. Future versions of the caching plugin should contain knobs to offer fine grained control over these features.

The proxy server requires at least one block of RAM for each file. Each block is reused after it has been written to disk. Note that RAM usage grows with the number of open files: *N_read*blocksize,* plus *N_prefetch*blocksize* if prefetching is activated. With sufficient server memory all clients are granted equal access to files. Obviously workloads that vary significantly would benefit from dynamically adjusting how blocks are allocated. Thus one could have a pool of extra blocks available, managed by a plugin. For lightly loaded servers the result would be quicker delivery of data to clients, while heavily loaded servers would benefit from the better memory management.

Caching proxy server load management: Typically, the cache disk cannot process all write requests before the write queue is filled. When the write queue is full, read requests not already present in the cache are handled as normal read calls, i.e. no extra buffers are allocated and the data is not cached. Note that the caching proxy will always serve read requests when data is already on disk. This overloading impact can be mitigated by optimizing the local disk configuration. Obviously this depends on the local disk device used. For the simple case of directly attached storage with disks managed through a commodity RAID controller, current testing indicates that using individual disks (binding them with the XRootD *oss.space* directive) outperforms a RAID-0 configuration by up to 30%. One can also reduce write affinity of the disk I/O scheduler, which is about a 10% effect. The performance improves with the number of disks available as expected. A rule of thumb is to use around 20 SATA disks per 10 Gbps of network I/O.

Caching proxy clusters: the typical environment where a caching proxy would be most useful is one in which network and disk I/O are likely to be fully saturated. This would be particularly true for larger Tier-3 clusters, Tier-2 centers allowing high levels of overflow jobs, and large scale opportunistic HPC clusters. The solution here is to build a cluster of proxy servers. This permits horizontal scaling that can be matched to the anticipated number of clients. Standard XRootD clustering is employed: a local redirector and set of caching proxy servers, each with directly attached highly performing disk arrays.

3. Testing

We describe two kinds of tests: systematic (scripted) testing from a Tier-3 cluster and tests with a Tier-2 analysis queue in production.

3.1. Systematic stress testing

As a Tier-3 like test, we ran 750 concurrent jobs from a SLAC Tier 3 cluster which has an aggregate 35 Gbps network connection. The cluster was dedicated to us during the stress test periods. We prepared a dataset of about 5000 files, all residing on disk across the country at the Tier-1 center at Brookhaven Laboratory in New York. The caching server was a Dell R730xd, Intel Xenon CPU E5-2643, 128 GB, Intel X540 10 Gbps NIC, H730 RAID controller, 12x2TB 7200 rpm disks configured as RAID-0. A block size of 1 MB was used, and the proxy caching server memory was restricted to 30 GB. Each job would do the following which emulates read access for a typical analysis job:

1. Randomly select a file from the dataset
2. Randomly determinate the number of reads from the file (0 - 64), the offset within the file (starting position) of each read, and the length of each read (0 - 32KB)
3. Order the above reads based on offset position
4. Data read from network are discarded and the next read follows immediately.
5. Go to 2 until the total number of reads exceeded 1024

6. Close the file and go to 1
7. All jobs started at about the same time, and ran with a wall time limit of 2.5 hours.

The first test was with a cold cache, followed by a warm cache test. Figure 3 depicts the egress and ingress network bandwidths for the two test cases. In these tests, we observed each job completing about 80 files in the 2.5hour time span, averaging about 250 MB/s read speed.

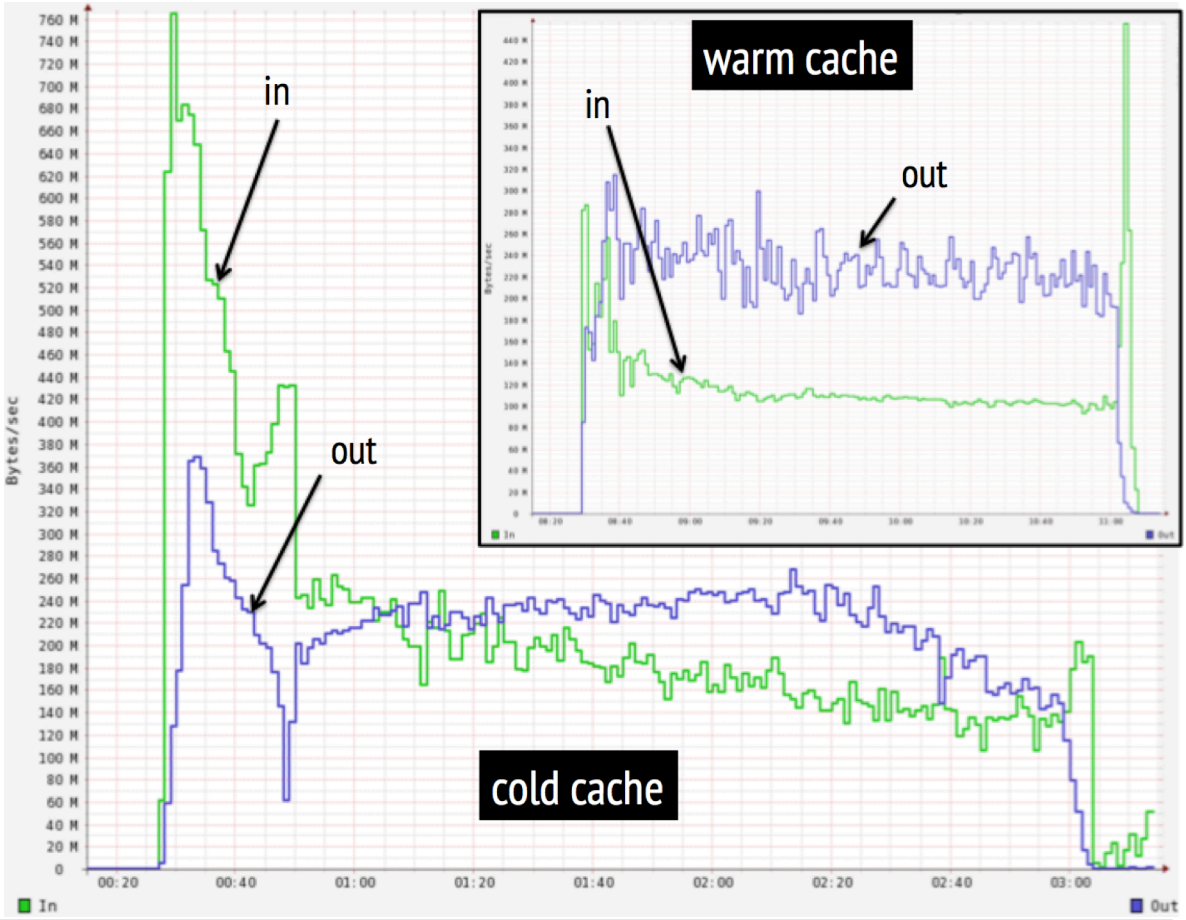


Figure 3. Network input and output I/O rate (MB/sec) for the stress tests of the caching server. In the plots, the vertical scales for the cold and warm cache tests range from 0-760 MB/s and 0-440 MB/s, respectively.

The caching behavior follows an interesting pattern. In the cold cache the WAN is taxed heavily at the beginning as new data is transported from remote storage. Data is quickly delivered to the cluster clients and plateaus at essentially the aggregate disk read speed, and less data is required of WAN as more data is read from local disk. For the warm cache case the initial taxing of the WAN is absent as local copies are available from the cache.

3.2. Load tests with ATLAS analysis jobs

To assess the XRootD proxy server potential for a Tier-2 center environment running realistic ATLAS analysis jobs, we deployed a server at the Midwest Tier-2 site in Chicago with the following characteristics: a Dell R730xd E5-2650 v3 @ 2.30GHz, 40 core, 96 GB RAM, Intel X540 10 Gbps NIC, connected to the (LHCONE peered) WAN via a Juniper EX9206 router at 8x10 Gbps. The storage setup is as follows: (10) 8TB disks in RAID-0, deadline scheduler selected for the kernel, *nr_requests* block tunable set to 1024 (I/O request queue size, default 128), 5120 KB read-ahead, disk cache policy enabled, using 2GB NVRAM on RAID controller, controller set to read ahead & write

back modes. The caching software was configured to run in file block caching mode with block size of 256kB. We allowed up to 48 GB of memory to be used by the cache software as buffer. When a requested data block was not found in the cache, it was read from other Tier-1 or Tier-2 center via FAX. The standard ATLAS PanDA "analysis queues" were configured to redirect I/O through the proxy cache, and "overflow jobs" from another center were directed to the site.

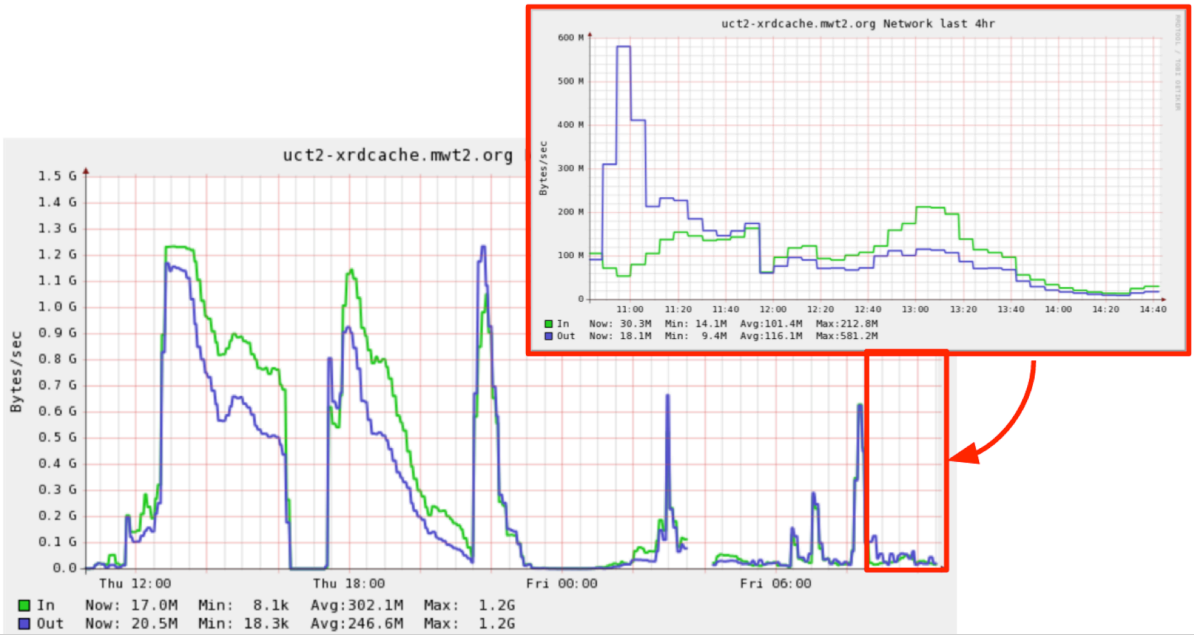


Figure 4. Network input and output I/O rate (MB/sec) for realistic Tier-2 tests of the caching server. For each, the green curve is traffic flowing into the proxy caching server, the blue for data flowing from the server.

The proxy behavior in this context is driven by an actual flux of overflow jobs during a 24hour production period, leading to uncontrolled peaks and periods of inactivity. Typical of cold-cache behavior, a batch of clients from a set of jobs simultaneously request data through the cache: more data taken from the WAN than read from disk (green curves, "In") for these cache misses. Job sets demonstrating cache hits are shown in the red-boxed region; a set of clients read data from local disk, in this case relieving the wide area network from additional traffic. Realistic Tier-2 analysis workloads for ATLAS are varied as large numbers of users from many physics working groups consume the resources. It is clear our initial testing the benefits for Tier-2 analysis were limited as fewer files were repeatedly accessed. More likely pileup and event mixing would benefit more, which we plan to investigate using opportunistic production resources having no dedicated local storage.

4. Conclusions

We have explored using XRootD caching servers operating as either whole file caches or block level caches and in both the LHC Tier-3 or Tier-2 context. We observed stable performance of the XRootD proxy server with caching plugin for both controlled testing for at the level of overflow brokering at a large Tier-2 center and interesting behavior in terms of network I/O. For a Tier-3 cluster with limited local storage the benefits of a caching server are evident: one can achieve good read performance while reducing the need for local storage of custodial data sets. In the Tier-2 context there is no large benefit to be obtained by caching due to the large numbers of users and widely varying input data sets required. Smaller Tier-2 production sites having no local storage could benefit from caching.

References

[1] ATLAS Collaboration, The ATLAS Experiment at the CERN Large Hadron Collider, JINST 3 13617 (2008) S08003

[2] Serfon C, et al., Rucio, the next-generation Data Management system in ATLAS, Nucl. Part. Phys. Proc. **273-275** (969-975) [10.1016/j.nuclphysbps.2015.09.151]
[3] Maeno T, PanDA: distributed production and distributed analysis system for ATLAS, Journal of Physics: Conference Series **0119** (062036) 006 [iopscience.iop.org/1742-6596/119/6/062036]
[4] Gardner R et al. 2013 Data federation strategies for ATLAS using XRootD, 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2013), Amsterdam, The Netherlands [10.1088/1742-6596/513/4/042049]
[5] Dorigo A, Elmer P, Furano F, 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), Miroslav Husak and Nikos Mastorakis (Eds.). World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, Article 46
[6] Bauerdick LAT et al., 2014 XRootD, disk-based, caching proxy for optimization of data access, data placement and data replication, J. Phys. Conf. Ser. **513**, 042044