XRootD and Object Store: A new paradigm

Katy Ellis[1,*], *Chris* Brew[1], *George* Patargias[1], *Tim* Adye[1], *Rob* Appleyard[1], *Alastair* Dewhurst[1] and *Ian* Johnson[1]

[1]STFC - Rutherford Appleton Lab, Harwell, UK

Abstract. The XRootD software framework is essential for data access at WLCG sites. The WLCG community is exploring and expanding XRootD functionality. This presents a particular challenge at the RAL Tier-1 as the Echo storage service is a Ceph based Erasure Coded object store. External access to Echo uses gateway machines which run GridFTP and caching servers. Local jobs access Echo via caches on every worker node, but it is clear there are inefficiencies in the system. Remote jobs also access data via XRootD on Echo. For CMS jobs this is via the AAA service. ATLAS, who are consolidating their storage at fewer sites, are increasingly accessing job input data remotely. This paper describes the continuing work to optimise both local and remote data access by testing different caching methods.

1 Introduction

The UK Tier 1 is situated at the Rutherford Appleton Laboratory (RAL) and supports all LHC experiments, as well as a growing number of others in HEP, Astronomy and Space Science. The RAL disk storage system, known as Echo [1], is an Erasure Coded Ceph Object Store [2] with Grid compatible APIs. A description of Echo can be found in the next Section. Access to Echo is primarily via XRootD [3], which is both a protocol name and an open source suite of fast and highly scalable data access tools commonly used by LHC experiments. However, XRootD is designed to use with file systems organised in directories.

Echo is not a file system in the traditional sense and does not use a directory structure. Instead it contains *objects*, which can be named so as to keep the appearance of a directory structure, e.g. `/my/path/isActually/anObjectName.root`. The entire *path* is a string which defines the object name.

2 The Echo Storage System

Echo is based on Ceph technology. At the core of Ceph is the *CRUSH* (Controlled Replication Under Scalable Hashing) *map* which is a description of the storage infrastructure and is used to assign data to particular storage locations whilst ensuring the required level of data redundancy. The Echo cluster is made up of multiple pools which provide storage to a single large, or multiple small experiments. Each pool has multiple associated *Placement Groups*

[*] Corresponding author: katy.ellis@stfc.ac.uk

which are each composed of a set of hard disks. Each disk is managed by an Object Storage Daemon (OSD). The Placement Groups and OSDs have a many-to-many relationship.

When writing data to Echo, the objects to be stored are first divided into 64 MB *stripes*. The software that performs this is known as the *libradosstriper* [4] and is one of the standard Ceph APIs. Each stripe is assigned a Placement Group via a hash algorithm. Each stripe is split into 8 MB *shards*. Three additional *parity* shards are calculated and all eleven are stored on different storage nodes. Of these, any eight are required to reconstruct a stripe. Three out of eleven OSDs within a group can be unavailable and the data will remain fully accessible. If up to three shards are lost, the system will recalculate the missing shards and 'self-heal'.

This type of storage does not use a central database for metadata look-up, and hence scales extremely well. Each client accesses Echo directly via a *gateway* which has the ability to both write and read data as described here. A gateway can be configured as a lightweight service and is installed on every local worker node.

Although the Echo object store provides several advantages, there are also some disadvantages to bringing this new technology into existing experiment analysis models. RAL is unique among WLCG Tier 1s in incorporating this technology although some Tier 2s are employing a similar setup, for example Glasgow in the UK. One of the challenges encountered during Echo development is the optimisation of access to Echo via XRootD for the different use cases. Several of these are described in this paper.

3 XRootD setup at RAL

XRootD is used for both file transfers in and out of Echo from offsite and for jobs running on RAL's batch farm to access local disk storage. There is also a *gridftp* plugin for transfers between sites, but the intention is to phase this out in the medium-term and replace the functionality with both WebDav (HTTP) and XRootD protocols [5].

The Third Party Copy (TPC) functionality was previously present in XRootD, however it was not commonly used due to incompatibilities between storage systems. In 2019 a substantial cross-collaboration effort was made to put it into operation [5, 6]. A TPC transfer copies a file directly from the source to the destination without streaming it via the command issuing machine.

As described in Section 2, access to Echo must be done via a gateway. These can be found on each individual worker node as light weight software layers, as well as on dedicated, heavy weight, externally facing machines for site-to-site transfers, which are known as *external gateways*. Supported specifically for the CMS experiment is the *AAA* service, which stands for *Any data, Any time, Any where* [7, 8] and must also run a gateway. Figure 1 illustrates these examples schematically, including interfaces between XRootD and libradosstriper.

On top of the Ceph gateway plugin, different flavours of cache or other access mechanism may be placed. XRootD caching at RAL is currently set up differently for each type of gateway, with disk caches on the worker nodes, memory caches on the external gateways and no cache on the AAA proxies. Previous investigations into cache optimisation were presented in [9].

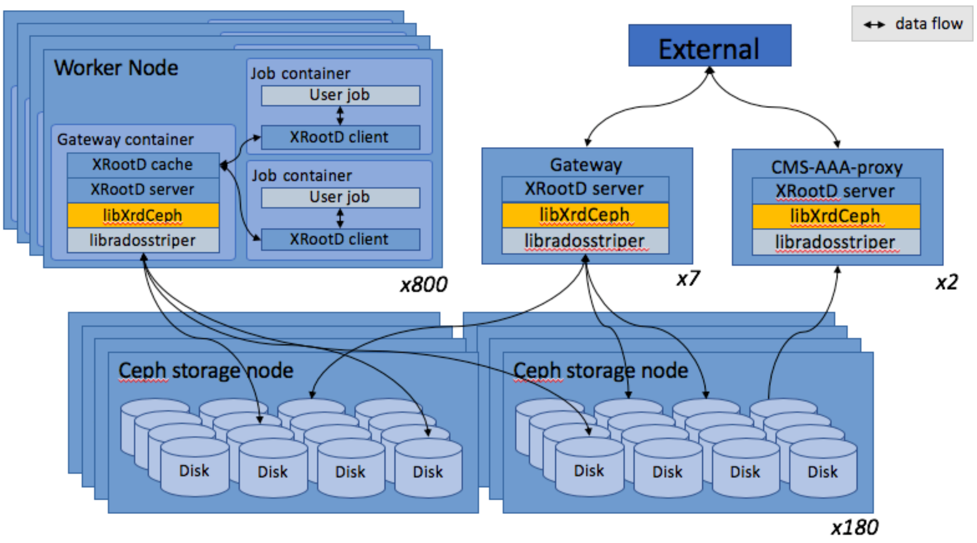
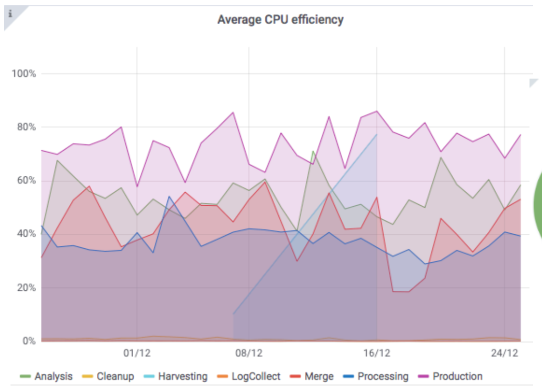
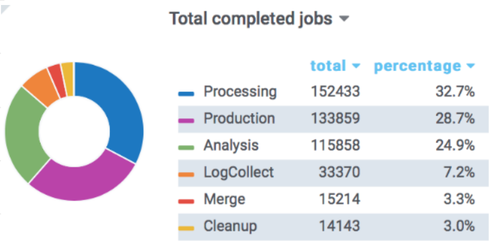


Figure 1: XRootD interactions with Echo

4 CMS Jobs

RAL has observed that CMS jobs spend a significant amount of time in I/O wait and this reduces their efficiency. Understanding how these jobs access their data is therefore key to improving their efficiency. Figure 2a shows the efficiency (CPU / Wall Time) of different types of CMS jobs run. Figure 2b shows the breakdown of the types of jobs run at RAL. It is therefore expected that the biggest gains in this area can be achieved by improving jobs of type ‘Processing’ as these jobs are most commonly run at RAL, and have among the lowest average efficiency. They typically have high I/O requirements, and stream multiple files of size ~4 GB from local storage. If the data is not immediately available via local storage, the AAA service is queried, and the files or parts of the files may be transferred from any other CMS site worldwide.





	total	percentage
Processing	152433	32.7%
Production	133859	28.7%
Analysis	115858	24.9%
LogCollect	33370	7.2%
Merge	15214	3.3%
Cleanup	14143	3.0%

Figure 2a: The average CPU efficiency of RAL jobs

Figure 2b - the number of each job type

CMS differs in data access methods from some other experiments which run jobs at RAL. Where others may download an entire file before it is needed, CMS keeps an open connection to the storage and accesses only the parts of the file as required. The advantage of this is that less data overall may be transferred to the worker node. However, the particular configuration makes multiple small reads less efficient than reading the whole file and further optimisation could be possible. In addition, studies such as [10] have shown that changes made at the level of the CMS software have had a significant effect on the performance of disparate storage systems.

4.1 CMS jobs and Echo – test setup

A typical RAL worker node with specification as shown in Figure 3, was drained and isolated from the batch farm. The same node was used for every test. Jobs were submitted via 'condor_submit' [11] to replicate the normal job submission method, and run in the full containerized structure as shown in the Figure 3 schematic diagram.

Hostname	lcg2069
Memory	131 GB
# CPU	32
Network b/width	1 GB/s
Storage type	Hard disk
Disk volume	1800 GB

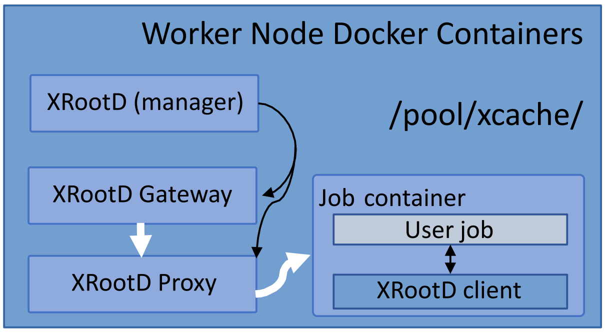


Figure 3: Specification and schematic of RAL worker node setup

The tests described in this Section use an adapted analysis job to do some event selection and write a ROOT output file of event properties. The job has to access multiple files but is not particularly CPU-intensive. A particular dataset was selected for the job to take as input, with greater or fewer files used for different tests. The dataset was placed on Echo in the /tmp/ area so that the files would always be accessed from local storage and never be deleted by the CMS data management system.

4.2 CMS jobs and Echo – cache hint

CMS jobs have an additional line in the XRootD configuration with name *cache-hint*. A complete description of this can be found in [3]. As described in Section 3, the way CMS jobs access data is not as straightforward as simply downloading entire files to the worker node. The cache-hint allows some individual configuration for non-homogeneous storage systems. The current standard at RAL is termed *Lazy-download*. This option caches the file on to the job in chunks of 128 MB at a time, and was thought to mesh well with the Echo stripe size of 64 MB. The test described in this section attempts to demonstrate whether this really is the best option.

The other cache-hint values are *Application-only*, where ROOT does the caching, and *Storage-only*, in which the caching is handled by the layer just below the CMS software framework. There is also an *Auto-detect* option where XrootD makes the decision itself based on the storage type. Results indicate that this is selecting *Application-only* for the RAL worker node.

Approximately ten test jobs were run for each cache-hint. Each job had 4 files as input, with a total size of ~9 GB. Table 1 shows the mean time taken to run the job, the standard deviation, and the total volume transferred by the job.

In this test, *Lazy-download* has transferred the entire file, whereas other options have cached a maximum of 75% of the data. The quickest option, *Application-only,* has a mean time of approximately 90% of the mean time taken by *Lazy-Download*, which was the slowest on average. Reported CPU (or 'User') time was very similar for all tests.

Table 1. Results from the cache-hint tests, averaged over ~10 tests per cache-hint.

Cache-hint	Mean time, min:sec	Standard deviation, min:sec	NetworkInput, Mb
Application-only	32:11	0:27	~6900
Lazy-download	36:04	0:26	9173
Storage-only	35:15	0:35	~6680
Auto-detect	32:13	0:31	~6900

Results indicate that it is worth attempting larger scale tests with the cache-hint options and particularly *Application-only*. These small-scale tests are not conclusive, as CMS jobs are of various types and only one simple job has been demonstrated here. Reliability at production scale is also a concern.

4.3 CMS jobs and Echo – parallelization

Another suspected influence on the I/O efficiency of CMS jobs is the number of concurrent jobs or reads running on the worker node. The tests in this section process the same number of unique files in each case.

Employing the same isolated computing node described in section 4.1, jobs were run in batches as illustrated in Table 2. Up to 16 jobs could be run in parallel with the original memory requirement of 8 GiB per job.

Table 2. Tests of increased parallelization, with mean averages over multiple runs where applicable.

# jobs	# files per job	Total job run time (min)	Time per file (min)	Average throughput per job (MB/s)	Total throughput (MB/s)
1	32	278	8.7	4.42	4.4
2	16	141	8.8	4.42	8.8
4	8	83	10.4	3.87	15.5
8	4	51	12.8	3.24	25.9
16	2	43	21.5	2.14	34.3

Results in Table 2 show that there are substantial time savings to be made for this job type with extreme parallelization. All 32 unique files were processed in 36 minutes when maximally parallelized compared with 278 minutes when all of the resource was used by a single job. However, note that the 'merge' part of the job, where results are combined into a single output has not been performed in this test.

The average throughput per job indicates that there is some slowdown in data access as the number of simultaneous reads increases above eight, as indicated by Figure 4.

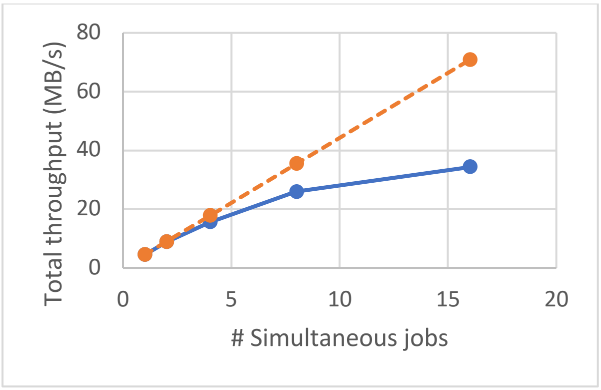


Figure 4: Throughput achieved with increasing number of simultaneous jobs (solid line), compared with perfect scaling (dashed line).

5 CMS AAA

The CMS AAA service is designed to provide partial or full datasets to any CMS User without the need to download it to local storage. This particularly allows for greater resilience against damaged or missing input files.

CMS jobs commonly read only a small fraction of any file. When Echo was initially configured there was a concern that many remote reads of small amounts of data could put significant load on the storage. The CMS AAA servers were therefore configured with memory caches. These caches fetched a minimum of 32 MB of data at a time, which was expected to reduce the load on Echo. Figure 5 shows the network throughput into and out of the CMS AAA proxies. The much higher input shows that most of the pre-fetched data was never used. The result was an overloaded service that required frequent intervention.

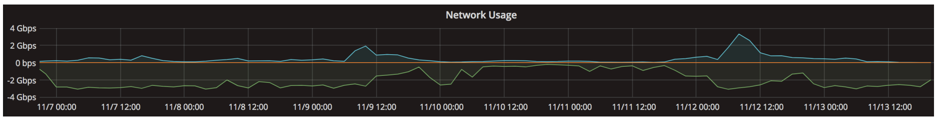


Figure 5: Network Usage over one week for the CMS AAA proxy before changes to the caching. Inflowing data to the proxy is shown as negative and outflowing data as positive. The scale is between -4 Gbps and 4 Gbps.

The solution was to simply turn off the memory cache on the proxies. The service then became much more stable and rarely required intervention. The AAA proxy Network Usage became balanced with respect to inflowing and outflowing data, as can be seen in Figure 6.

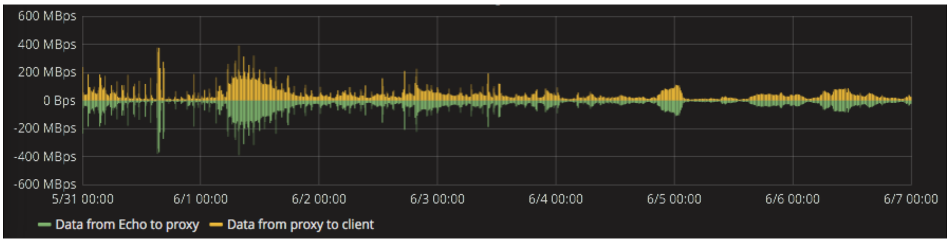


Figure 6: Network Usage over one week for one CMS AAA proxy after changes to the caching. Inflowing data to the proxy is shown as negative and outflowing data as positive.

6 Conclusions

It has been shown that file transfer times out of the Echo storage system are sensitive to changes in cache settings. The CMS AAA service was made significantly more stable by entirely removing the proxy cache. Jobs on RAL worker nodes were shown to be around 10% quicker when using alternate cache-hint values in place of lazy-download. Multiple, concurrent reads from the same worker node showed some slowdown in the average data rate per file, and this effect was most significant for eight or more simultaneous reads. This could indicate that the bottleneck is on the worker node and requires further investigation.

References

1. A. Dewhurst et al., J. Phys.: Conf. Ser. **898** 062051 (2017)
2. The Ceph website: https://ceph.io
3. The XRootD website: https://xrootd.slac.stanford.edu
4. The Ceph architecture : https://docs.ceph.com/docs/master/architecture/
5. A. Forti et al., *Modernising Third-Party-Copy Transfers in WLCG*, presented at CHEP 2019, 10.5281/zenodo.3599529
6. W. Yang et al., *Xrootd Third Party Copy for the WLCG and HL-LHC*, presented at CHEP 2019, 10.5281/zenodo.3599613
7. Description of CMS AAA: https://twiki.cern.ch/twiki/bin/view/Main/CmsXrootdArchitecture
8. L. Bauerdick et al., J. Phys.: Conf. Ser. **396** 042009 (2012)
9. A. Dewhurst, A. Lahiff, poster presented at CHEP 2018: https://indico.cern.ch/event/587955/contributions/2936908/attachments/1683145/2705081/CHEP18XrootD.pdf
10. Leonardo Sala et al., J. Phys.: Conf. Ser **331** 072062 (2011)
11. The HTCondor manual: https://research.cs.wisc.edu/htcondor/manual/