Experiences and developments of the RAL-LCG2 Tier-1 object store ECHO in Run-3 and preparing for HL-LHC

J Walder[1,*], *T* Byrne[1], *A* Dewhurst[1], *I* Johnson[1], *A* Rogovskiy[1], and *J* Thomas[1]

[1]STFC Rutherford Appleton Lab, Harwell, UK

Abstract. Data storage at the UK Tier-1 facility at RAL is provided through its ECHO storage, serving the requirements for the WLGC and increasing numbers of other HEP and astronomy related communities. ECHO is a Ceph-backed erasure-coded object store, currently providing in excess of 40PB of usable space, with frontend access to data provided via XRootD or GridFTP, using the libradosstriper library of Ceph. The storage must service the needs of: high-throughput compute, with staged and direct file access passing through an XCache on each workernode; data access to compute running at storageless satellite sites; and, managed inter-site data transfers using the recently adopted HTTPS protocol (via WebDav), which includes multi-hop data transfers to and from RAL's newly commissioned CTA tape endpoint. A review of the experiences of providing data access via an object store within these data workflows is presented, including the details of the improvements necessary for the transition to WebDav, used for most inter-site data movements, and enhancements for direct-IO file access, where the development and optimisation of buffering and vector read strategies is explored.

1 Introduction

The Rutherford Appleton Laboratory (RAL) plays a crucial role in supporting the data-intensive requirements of the Worldwide LHC Computing Grid (WLCG). To facilitate efficient data access and storage for high-energy physics experiments, RAL has established the ECHO [1, 2] infrastructure. This infrastructure combines the XRootD [3] and Ceph [4] technologies to provide a robust and scalable solution for handling vast amounts of data.

One of the primary use cases is the distribution of experimental data generated at various Large Hadron Collider (LHC) experiments, i.e. ALICE, ATLAS, CMS and LHCb. The ECHO cluster employs XRootD as a high-performance data interface allowing researchers worldwide to efficiently retrieve and replicate these datasets for analysis. To enhance data redundancy and reliability, the ECHO infrastructure leverages Ceph's distributed storage capabilities. This redundancy ensures that data remains accessible, even in the face of hardware failures or other unforeseen issues.

In preparation for Run-3 of the LHC, and to prepare for the requirements of HL-LHC, a number of optimisations and updates have occured since the original deployement of ECHO, such as the migration [5] from GridFTP [6] to WebDav [7]. A review of the changes to ECHO to meet these challenges that have been adapted so far, focusing on XRootD, is provided in this note.

*e-mail: james.walder@stfc.ac.uk

2 ECHO and the Ceph storage cluster

ECHO is built on the Ceph distributed storage platform, which offers unified and scalable storage capabilities. Ceph uses a distributed architecture consisting of storage nodes, known as Object Storage Devices (OSDs), to store data. These OSDs are distributed across the cluster for load balancing and redundancy.

To optimise storage efficiency, ECHO employs erasure coding (EC) techniques. This method reduces data replication overhead while maintaining data integrity and availability. Erasure coding is particularly useful for large-scale, exabyte-sized storage systems like ECHO. For the data pools currently used by the major experiments, 8 + 3 EC is used.

At the time of writing, approximately 80 PB of raw storage is deployed for use, with over 6000 OSDs across approximately 300 Storage Nodes (SN). A host level failure domain is used, such that the distribution of data in placement groups across the OSD means that failure of any single SN will not risk the integrity of the data.

3 XRootD and data access

While RAL and its ceph storage provides for CephFS, S3 and SWIFT endpoints, the data stored for the experiments uses the ceph as an Object Store, interacting at the rados layer using functionality provided by the libradosstriper [8] and *XrdCeph* [8] plugin to XRootD (described below). The libradosstriper library adds a lightweight layer on top of rados interface, allowing a file to be split into individual ceph objects with configurable object and stripe sizes. While rados continues to manage the direct operations against ceph, libradosstriper adds the additional functionality to manage the spliting and striping of data across these objects. libradosstriper provides for mostly atomicaly correct behaviour, however, and as will be discussed below, due its locking and unlocking behaviour, the performance against small reads is not optimal. In the case of high-energy physics applications, the majority of use-cases are Write Once, Read Many (WORM) requests, and hence there is the scope for improved efficiency under these scenarios.

In practice, with the implementation of ECHO, libradosstriper is configured such that a file writen to echo is divided into 64 MiB ceph objects, with names suffixed with a 16-digit 0-padded hex number (incrementing by one for each object, separated by a dot to the original file name) before being written into Ceph, and with the EC applied. Figure 1 illustrates the way a typcial file would be stored on ECHO.

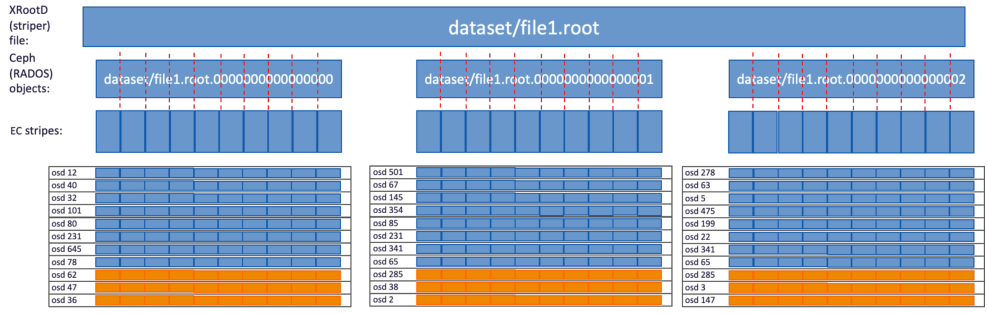

Figure 1. For a typical file on ECHO, the libradosstriper divides the file into (a configurable) 64 MiB objects, with each object then being striped and erasure coded across the OSDs of the placement groups.

In Fig. 2, the distribution of data across OSDs and SNs is shown. In this example, a 10 GiB file is divided into 157 ceph objects, which are placed over 1400 OSDs on approximately 230 SNs. i.e. data occupies approximately 6 OSDs per SN and correspondingly, a large file is likely to be distributed across most of the SNs in the ceph cluster.

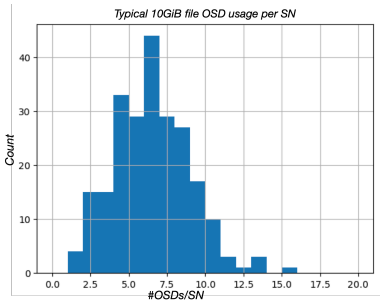

Figure 2. For a 10 GiB file, a typical distribution of the number of OSDs that data is distibuted across per storage node.

While Ceph provides the storage backend, an interface is required for clients to interact with it. XRootD [3] provides a lightweight and flexible framework that satifies the needs for WLCG and other communities. In addition, the XRootD plugin style design allowed for the development of the XrdCeph plugin [8] (using libradosstriper) to bridge between XRootD and the rados layer of Ceph. XrdCeph inherits from the *XrdOss* layer to provide this interface, presenting a restricted set of access operations. For example, as an object store has no direct concept of directory hierarchy, and as a listing over all objects in a (large) ceph pool is prohibitly expensive, no possibly to list files is available. Similarly, as a rename operation is effectively a copy with a new name, resulting in movement of data across to new OSDs and placement groups, this is also a feature that is not supported.

4 Improving performance for Run-3 and beyond

With the evolution to Run-3, which includes improvements and updates in other layers of the overall grid middleware, ECHO (including XrdCeph) has consequently needed to adapt to these changes. These changes include the deprication of GridFTP, in favour of WebDAV, resulting in signficant usage of XrdCeph compared to the GridFTP implementation [1, 9]. For transfers using the xroot protocol, XRootD now uses *paged* reads and writes. This is a change that alters the reads and writes that were in (typcially) 8 MiB blocks, to reads and writes of size < 100 kB. The motivation here from the developers is to add additional inflight checksumming to identify and resend corrupted data at the small chunk-sized level, rather than to wait for the checksum of the final transfer to discover potential data corruption. In this case, with the design of libradosstriper and XrdCeph, small reads and writes are not efficient against Ceph, due to its locking and unlocking behaviour.

Vector read operations[1] were not supported in the initial implementation of the XrdCeph plugin. As a consequence, each vector read operation resulted in unfolding each of the batched set of reads into individual read requests, and hence suffered from significant overheads.

[1] Vector reads are typically used in direct access from jobs running on worker nodes to access directly only the bytes of data that are useful for the client, rather than downloading the entire file. They are typically a single set of batched byte ranges to be read.

To overcome these challenges and to continue to improve the throughput for Echo, two strategies were deployed. Firstly, a simple buffering layer in XrdCeph was deployed, to reduce the number of reads and writes made against Ceph (using the libradosstriper). For reads, if the data for a requested byte range was not already not in the buffer, a request to ceph is issued for a single read (the size of the buffer). Further contiguous reads for data that exists within the buffer is then the returned directly from the buffer. Similarly, for writes, data is written into the buffer, and once full, a single write is made to Ceph via libradosstriper, ensuring that small writes are not propagated through.

Secondly, for reads, it would be possible to bypass the locking and unlocking behaviour in XrdCeph, where significant overhead on small reads can be introduced, and additionally implementing efficient vector read support at the rados layer [10] (rather than in libradosstriper).

4.1 Buffering in XrdCeph

The performance of introducing a buffer into the XrdCeph layer is showin in Fig. 3. For davs, reads and writes are typically requested (within the XRootD framework) in 1 MiB blocks. For root protocol transfers, these requests can range from approximately 64 kB for paged requests to 8 MiB. The plots show the performance for single file transfers in a development cluster environment. In production, a buffer size of 16 MiB was empirically found to optimise throughput and memory usage. This approach is fully efficient (in terms of data read from Ceph) for cases where the whole file is read. In cases where on small and sparse sections of the file is read, this appoach can lead to inefficiencies in reading uncessary data. With this buffering, no attempt was made to buffer explicit vector read requests for this reason.

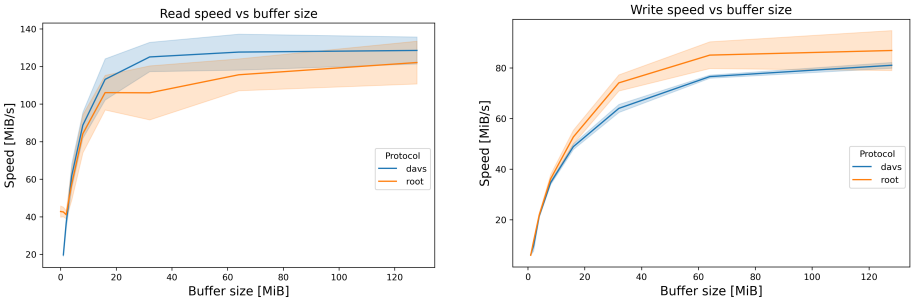

Figure 3. Performance of read (left) and write (right) speed for different sizes of XrdCeph buffer, measured using root and davs protocols for a 1 GiB file. The lines represent the mean of the distribution for each buffer size, and the colour bands indicate the statistical uncertainty.

4.2 Striperless reads and read vector implementation

The second strategy employed to improve throughput for both reads and vector reads (write requests are not altered by these changes) was to avoid the use of the libradosstriper interface – removing the locking and unlocking steps – and to implement efficient vector read support in XrdCeph, using rados directly. Technicaly, some aspects of libradosstriper are reimplemented, such as the need to map between a read request's byte offset and length, and map this to the corresponding local offset and read length (noting that a read could cover multiple ceph objects) of the (at least one) ceph object of the data written via libradosstriper. The

removal of locking for reads has very limited practical concerns, given the WORM nature of operations.

In Fig. 4 the read data taken from Fig. 3 is shown, together with data taken using the non-striper reads, for differing sizes of buffer. While the data suggests that the non-striper implementation is most performant with no – or very small – buffer sizes, when tested in the production environment, the large number of small reads could still have a negative impact on the performance of the buffer, hence it is still useful to keep some degree of buffering.

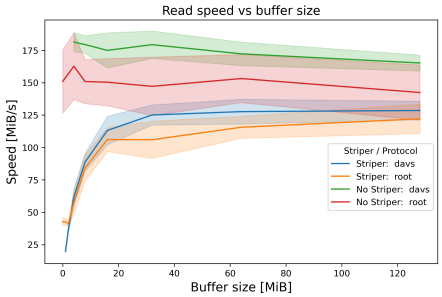

Figure 4. Performance of read speeds for different sizes of XrdCeph buffer, measured using root and davs protocols for a 1 GiB file. In addition to the curves overlaid from Fig. 3, the effect of reads without the libradosstriper are shown, also for various sizes of buffer.

For the vector reads, the set of reads (described by a list of offsets and read-lengths) are mapped to their ceph objects, and a local set of offset and lengths. These sets are then issued to ceph via librados in a batched operation. This replaces the default operation that converts each chunk of the vector read into a single read, into a small set of batched read requests. Figure 5 compares the performance between the default implementation and the updated reads. The test procedure is as follows:

- For each test 32 processes are submitted simultaneously;
- Each process connects to the storage, executes 100 readv requests, then disconnects; the procedure is repeated 10 times (i.e. each process submits 1000 readvs overall);
- Each individual readv request has 900 chunks;
- Chunks are distributed over 42 MiB;
- The length of every chunk is distributed between 1 and 1024 bytes.

The improvements are visible through both the lack of failed reads, and improvements in the mean and meadian read times from 6.0 s to 1.1 s and from 0.84 s to 0.73 s, respectively.

5 Improving metadata operations for checksum requests

A request for a checksum (typically using the adler32 algorithm) can proceed in one of two ways (assuming a valid file, etc.). Firstly, if no checksum has previously been computed the file is read back to the XRootD gateway and the checksum computed. This computed checksum is stored in ceph as metadata on the first object of the file, and the resulting checksum returned to the client. The time taken for this calculation is typically O(10) s/GiB. Secondly, if there is a checksum already computed against the file, then the checksum is retreived from the object metadata and returned to the client.

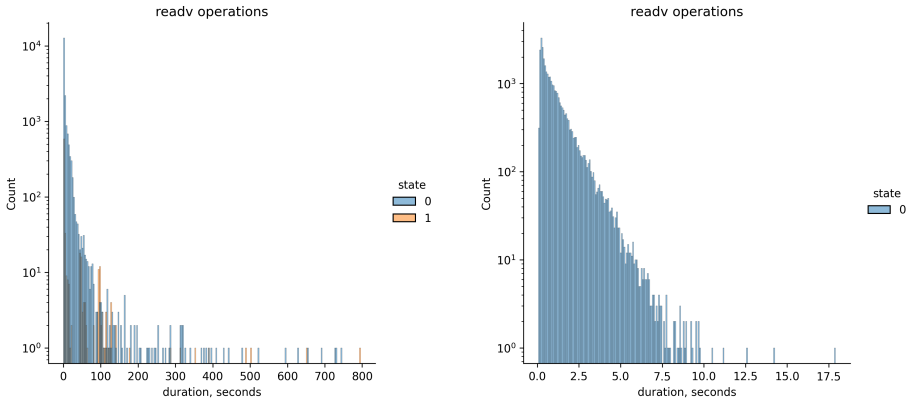

Figure 5. Performance of vector read operations on the original (left) and update (right) XrdCeph plugin. State 0 and 1 represent successful and failed reads, respectively. The horizontal axis represents the duration of each operation, noting the difference in scale between the two plots.

In ECHO, this is currently technically achieved via a call to an external python program, which connects to Ceph via the rados client python bindings, performs the necessary operations and then disconnects from the cluster. While this is acceptable for cases where the checksum needs to be computed from the file, the setup and teardown of the connection to the cluster can introduce a significant overhead, when the checksum need only be retreived from the metadata. To improve this, a client-server model is under development. A service runs on each gateway host that already holds the connection to the ceph cluster, the client - the external application called by XRootD - merely connects to the local server and request the checksum for the given file.

In Fig. 6 a comparison between the current and development implementation is shown, for the case of checksum retrieval from metatadata only. Other development options for checksumming are also being considered, such as implementing checksuming with the XRootD plugin framework, or, to implement on-the-fly checksumming via the HTTP plugins of XRootD, which would avoid the necessity to re-read the data post initial write. A more recent understanding of the XRootD code has highlighted a difference in scheduling of checksum requests when an external program is used, compared to when the internal XRootD checksum calls are used, the later potentially being scheduled immediately (in the case of a checksum already stored in the metadata).

6 Resiliance and failover

A DNS round-robin was implemented (at the time of the conference) to distribute the load across the XRootD servers (aka. Gateways) that act as the frontend to the ECHO cluster. While this is a simple and effective solution, it does not accommodate for failure of an individual hosts, or any active load balancing. An alternative approach is to use the CMSD service provide by the XRootD software to actively manage the redirection between a frontend manager and the server hosts. Figure 7 illustrates the current configuration that is expected to be deployed into production shortly. A DNS round-robin entry point to two virtual floating IPs. Using keepalived on two XRootD manager hosts to provide failover support in case one of the managers fails. Using the managers and CMSD, the client is redirected to an appropriate server to handle the actual transfer.

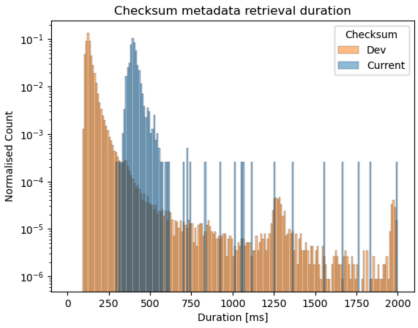

Figure 6. Comparison of checksum metadata retrieval times using the current implementation and the development client-server approach. Mean retrieval time is observed to improve from 450 ms to 150 ms.

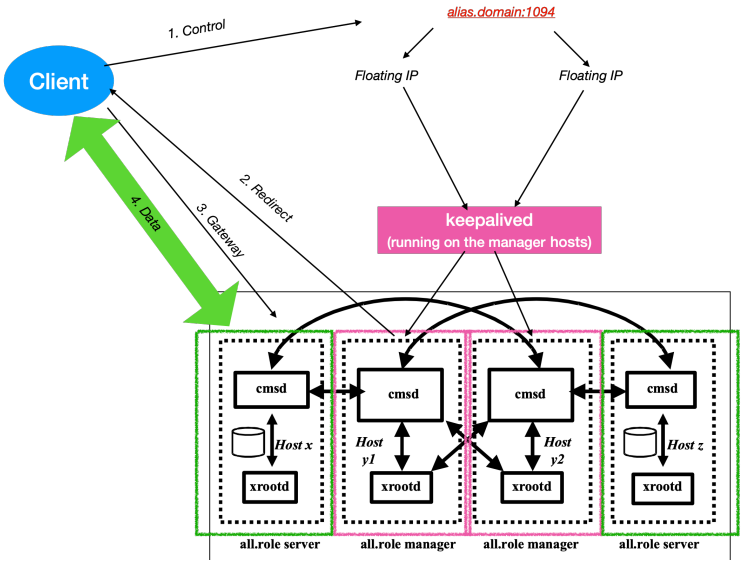

Figure 7. Illustration of clustered CMSD configuration using keepalived to provide manager failover. Adapted from Ref. [11]. Although the figure shows two servers, in practice over 10 servers are connected to the managers.

By transitioning from a DNS round-robin to actively managed CMSD setup it is hoped that better resiliance against a problematic cluster will be established, and that active loadbalancing will allow for an even distribution of throughput. Experience within the production environment will be needed to optimise the tuning of the CMSD parameters and to demonstrate these assertions.

7 Summary and outlook

In summary, the ECHO cluster, based at RAL, comprising of XRootD and Ceph, plays a pivotal role in supporting the data needs of the WLCG and other data-intensive organisations, enabling efficient data distribution and archive, and ensuring data reliability. The combina-

tion of XRootD and Ceph technologies provides a versatile and scalable solution that aligns with the demanding requirements of high-energy physics experiments currently at the LHC, and to scale to HL-LHC requirements. While scaling of the storage volume itself, and the horizontal scaling of the number of XRootD servers (acting as gateways) is necessary, it is demonstrated that improvements to the architecture and internal software are also required to provide an efficient, reliable, and cost effective solution. The improvements shown here for read, write and metadata operations, together with improved failover and loadbalancing support, are largely already in place for production opperations, and are demonstrating improved resiliance.

For the future, with increased operational experience of these recent changes, it is anticipated that there will be developments to the tuning of the CMSD, e.g. in the load-balancing parameters, to better spread load across the hosts, and to improve the ability of XRootD to failover bad gateway hosts. There are also ongoing studies on the performance of deletions. As deletions occur in realtime, rather than as an asynchronous or metadata process, the client waits for the successful deletion, which may take $O(s)$ or longer in some cases. It is important to charaterise the imapct on deletion rates at the levels needed for HL-LHC operations, and to understand whether this can be met with horizontal scaling of the number of Gateway hosts, or, if some further development will be required. With recent deeper understanding in the scheduling of checksum requests within XRootD, futher optimisations to the checksum operations may be possible and are under development. Lastly, the buffering layer has so far been used to mainly mitigate the overheads of small read or writes. Improved buffering may also offer significant opportunties to add a layer of latency 'hiding' between the backend storage to the client, and may also be a source of further performance optimisation.

ECHO already supports LHC and other experiments, and will continue to grow to ensure their future demands.

References

[1] A. Dewhurst, I. Johnson, J. Adams, B. Canning, G. Vasilakakos, A. Packer, *The deployment of a large scale object store at the RAL Tier-1*, in *Journal of Physics: Conference Series* (IOP Publishing, 2017), Vol. 898, p. 062051

[2] Ellis, Katy, Brew, Chris, Patargias, George, Adye, Tim, Appleyard, Rob, Dewhurst, Alastair, Johnson, Ian, EPJ Web Conf. **245**, 04006 (2020)

[3] *xrootd*, https://xrootd.slac.stanford.edu/index.html

[4] *ceph*, https://ceph.io

[5] B. Bockelman, A. Ceccanti, F. Furano, P. Millar, D. Litvintsev, A. Forti, EPJ Web of Conferences **245**, 04031 (2020)

[6] W. Allcock, *Gridftp: Protocol extensions to ftp for the grid* (2001)

[7] E. Whitehead, M. Wiggins, IEEE Internet Computing **2**, 34 (1998)

[8] S. Ponce, https://indico.cern.ch/event/330212/contributions/1718786/attachments/642384/883834/CephPluginForXroot.pdf

[9] https://github.com/stfc/gridFTPCephPlugin

[10] T. Byrne, *Optimizing Ceph IO for High Throughout Particle Physics Workflows*, https://static.sched.com/hosted_files/ceph2023/a4/Optimizing%20Ceph%20IO.pdf (2023)

[11] https://xrootd.slac.stanford.edu/doc/dev54/cms_config.pdf, Figure 1.1.1-2