Exploring XRootD Optimisations Using Advanced Monitoring in the UK

Robert Currie[1,*], *Wenlong* Yuan[1,**], and *Neofytos* Themistokleous[1]

[1]School of Physics and Astronomy, The University of Edinburgh, James Clerk Maxwell Building, Peter Guthrie Tait Road, Edinburgh, EH9 3FD

Abstract. XRootD servers are commonplace to many parts of HEP data management and are a key component to data access and management strategies in both the WLCG and OSG. Deployments of XRootD instances across the UK have demonstrated the versatility and expandability of this data management software. As we become more reliant on these services, there is a requirement to collect low-level metrics to monitor service performance and behaviour. This presentation will share our recent experiences in the collection, monitoring and analysis of such metrics from servers at UK WLCG-Tier2 sites.

Building on work presented at VCHEP-2021 we have been collecting XRootD service metrics from various UK sites. In addition to this, we have developed novel technologies for recording and verifying metrics from our grid storage systems at the Edinburgh Tier2. Our custom tooling is integrated with our recently deployed monitoring platform. This work complements ongoing WLCG efforts focussing on capturing and analysing XRootD based traffic flow. Through capturing and analysing additional metrics we're able to better remotely assess service health and provide insights to assist in debugging. The goal of this is to offer a centralised resource that monitors the performance and health of remote storage services across the UK.

Making use of these additional service metrics also allows us a greater insight into the performance of XRootD Proxy File Caches. By applying machine learning techniques to already collected metrics, we aim to determine if gains can potentially be made in optimising the behaviour of these caching systems in production at a UK WLCG-Tier2. The result of this potentially being the building and deployment of custom decision-making libraries for use with XRootD-PFC services.

1 Introduction

Within Tier2 sites hosted in the UK as part of GridPP we are working to better support remote data access for production WLCG workflows. To support this we plan to deploy XRootD [1] caching services at compute centric sites to facilitate accessing data at remote sites. In order to understand how best to configure/optimize such services, a monitoring system has been developed to collect metrics from a transparent XRootD-PFC (Proxy File Cache) instance at the Edinburgh Tier2 site [2].

*e-mail: rob.currie@ed.ac.uk

**e-mail: wenlong.yuan@ed.ac.uk

XRootD as a file server is capable of collecting, parsing and transmitting a large amount of file transfer metadata. Collecting and parsing this data allows for remote monitoring of file transfers. This monitoring data collected from XRootD as a service has proven to be challenging to handle correctly. However, it potentially offers a powerful insight into system and service performance, and potentially offers a way to measure the impact of efforts to tune and optimize the service and underlying storage system.

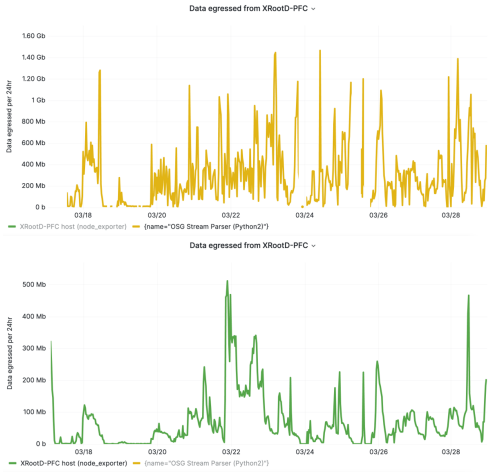

Figure 1. This plot shows recorded data throughput for an XRootD-PFC instance at Edinburgh.
Top plot shows the data throughput as reported by XRootD [1] monitoring messages.
Bottom plot shows the same data as collected by node_exporter [3].

As shown in Figure 1, the monitoring data collected from XRootD as a service suggests drastically different system performance to system metrics. The assumption for this difference is that the data collected from XRootD is correct, and that the problem potentially lay in the parsing and presentation of this data. In order to validate these metrics and perform a cross-check several tests were devised to be able to compare the data collected from XRootD with the data collected from the underlying storage system.

This testing framework also provides us with a setup which is able to be used to test the performance of the XRootD-PFC service as used in production. Through the use of these collected performance metrics we can potentially modify the cache behaviour to improve performance.

2 XRootD Monitoring

XRootD monitoring is built around the collection and parsing of metrics from the XRootD server itself. These metrics are broadcast in a binary format over UDP and a setup to collect and parse these metrics has been shown in previous work [2]. Parsing these metrics correctly requires an understanding of the data being collected and how it is being collected, as well as the format of the data being broadcast.

XRootD supports a number of different plugins and configurations. With that in mind, we are only insterested in testing an XRootD service instance which has loaded the XRootD-PFC plugin.

2.1 XRootD PFC Monitoring and .cinfo files

The XRootD-PFC plugin tracks the status and access history of files within the cache using ondisk metadatafiles stored alongside the cached data. These files are distinguished from data by having an additional .cinfo extension and contain information about the file, including metrics such as file-size, checksum, and the time of last access. These .cinfo files are updated when the file is accessed, and are created when the file is first cached. Information in these files is stored in a binary format with the layout summarised in Figure2.

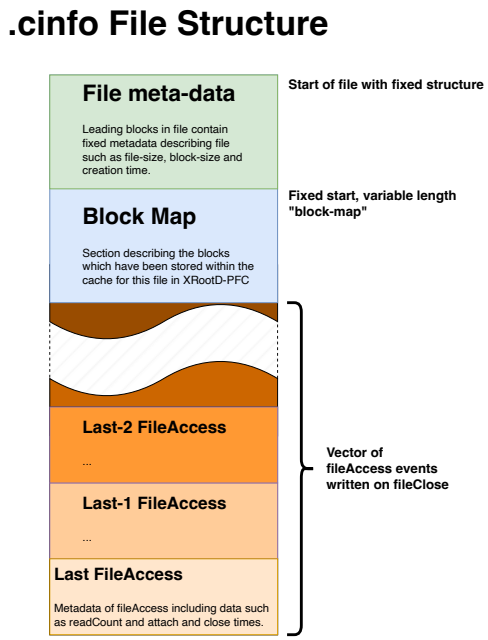

Figure 2. This diagram shows the layout of the XRootD-PFC .cinfo metadata file format as of XRootD-v5.6.2. [1]

The binary data in these files is similar to the broadcast monitoring data which means that a tool can be written to parse these files using similar code from the XRootD monitoring stack. At Edinburgh we decided to write a custom parser in Python3 with the intention of being able to parse the data in these files into a more human readable format such as JSON. This parser was verified as working with a standalone XRootD-PFC instance with single test file transfers.

3 XRootD-PFC Testing Methodology

Following on from previous work [2] our intention is to collect and store monitoring metrics from XRootD to be able to understand and optimize this service. As we're interested in understanding the performance of our XRootD-PFC in production we ran additional services in parallel to collect live metrics whilst the service was being used by production workflows.

Our testing primarily focussed on collection of metrics from a single XRootD-PFC inc-tance at the Edinburgh WLCG-Tier2.

This testing will focus on the amount of data ingressed and egressed from the XRootD-PFC service. As well as the collection of other metrics which may be useful for monitoring and optimization purposes.

3.1 XRootD-PFC Test System

Our XRootD-PFC test system has been deployed using an old Tier2 grid storage node. This system was a Dell R510 configured with 2xXeon E5620 CPUs and 32GB of RAM and a 10Gbps network connection. Our underlying storage was 12x1TB SAS HDD deployed using ZFS on a CentOS7 installation. The only significant change from a vanilla CentOS7 setup was to significantly increase the number of file handles and inotify watches available to services to be able to record file changes within our cache. Care has been taken when collecting performance metrics to ensure that the system does not become saturated or significantly overloaded by the workload imposed by the XRootD-PFC service.

This XRootD-PFC service was deployed as a transparent layer between the Tier2 worker nodes and the Tier2 storage using x509 authentication. A fraction of the production ATLAS workflow file access was routed through this service to provide a realistic test to collect performance metrics from.

This server was only used to run a XRootD-PFC instance and a Python3 performance monitoring service in parallel.

3.2 Monitoring Data collection

Messages from the XRootD service are broadcast over UDP and are collected by different services referred to as "stream parsers". This data is then preliminarily analyzed and metrics as stored in an ElasticSearch or OpenSearch cluster.

Collecting, parsing and analysing this data has proven to be challenging as it involves many different services. As a cross-check for errors in the monitoring stack, multiple parsers have been deployed in parallel to parse data from the same source. (IPtables is used to broadcast the same data to multiple endpoints) This allowed for cross-validation of the results collected from XRootD monitoring messages. In addition to this, data transfer rates are also compared to transfer rates extracted from the host using node_exporter [3] using prometheus [4].

3.3 XRootD monitoring parsers

Building on work presented at [2], 3 independent parsers have been deployed to offer cross-validation and verification of the results collected from XRootD monitoring components. These 3 parsers are, as follows:

- **OSG Stream Parser**
  Previously presented at [2] this parser is a modified form of a tool from the OSG [5] written to support additional metrics from the XRootD-PFC service. This parser has been written in Python2 and has been tested in production at Edinburgh.
- **RAL Stream Parser**
  This is a parser for data broadcast from XRootD which was written for internal use at the UK Tier1 RAL [6]. This parser was modified to support Python3 and is mainly deployed as a cross-check against the "OSG Stream Parser" as it relies on the same input data.

- **cinfo-extractor**
  This 'parser' in this service is a Python3 sentinal which parses .cinfo files directly on-disk from the XRootD-PFC service and broadcasts the data to a remote OpenSearch service.

Both the OSG and RAL stream parsers are designed to collect metrics broadcast over the network via UDP. By comparison the cinfo-extractor collects metrics from .cinfo files on-disk and then transmits them to a different endpoint using HTTPS. An agreement therefore between the 3 parsers would rule out any issues due to dropped UDP packets.

An agreement between the 3 parsers also means that data from .cinfo files are consistent with messages broadcast over the network. In this case if this data disagrees with node_exporter metrics it could point to a potential problem with the XRootD monitoring stack.

Ideally we aim to see an agreement between all 4 possible measurements of data ingress and egress from our test server.

3.4 cinfo-extractor

The Linux operating system provides a number of tools to monitor file changes. The main approach used in this study was to use the inotify-tools package. The cinfo-extractor tool makes use of the Watchdog library [7] in Python which allows for tracking of files on-disk via Linux inodes. This allows for the sentinal to be notified when a file is created, modified or deleted. By tracking events being written to disk during a cache update this sentinal is able to parse the .cinfo file and broadcast the changes in close to real-time.

When a .cinfo file was updated with new transfer information, the sentinal would re-parse the file and broadcast it's new content to a remote OpenSearch service. This OpenSearch service would then store this data in a database for later analysis. Using inotify it has been possible to identify files being accessed multiple times, file creation and file deletion.

3.5 Isolated testing

When further analysing and verifying components of our XRootD-PFC setup a 2nd test setup was deployed for testing purposes.

This system was a much simpler setup employing 2 lightweight XRootD instances in VMs, this allowed for tearing down and re-building the test setup to make tests reproducible. Data that was used in this test setup was generated using multiple entropy sources to generate large incompressible files. Between tests the VMs were torn down and rebuilt to ensure that the data was not cached in memory or on disk.

This testing will focus on repeatedly copying a single file from an XRootD source through an XRootD-PFC instance to a local posix filesystem.

4 Results

Testing the performance and behaviour of the XRootD-PFC and the associated monitoring stack ended up being separated into 3 different stages. Analysing file access patterns allowed us to examine the performance of the XRootD-PFC service to identify potential bottlenecks. Throughput testing allowed us to look at the performance and reproducibility of metrics collected from the XRootD monitoring stack. Isolated testing allowed us to verify the correctness of the data collected from the XRootD monitoring stack.

4.1 File Access Results

Figure 3 shows the total data transferred at fileClose as reported by XRootD monitoring messages vs length of time each file was open, grouped by file type. From analysing this and inspecting different workloads at our site we see that AOD files (blue in Fig. 3) are typically copied directly to worker nodes before being accessed. This matches the much higher data rates associated with their transfers vs DAOD files (orange in Fig. 3) which are typically accessed directly via the XRootD-PFC service.

With this in mind we suggest that a simple optimization for an XRootD-PFC moving forward when supporting the ATLAS workflow would be to prioritize the caching of DAOD files over AOD.

Whilst this is not a generic optimization we would be able to apply to all workflows, it may potentially optimize our cache performance by reducing the amount of data that needs to be cached.

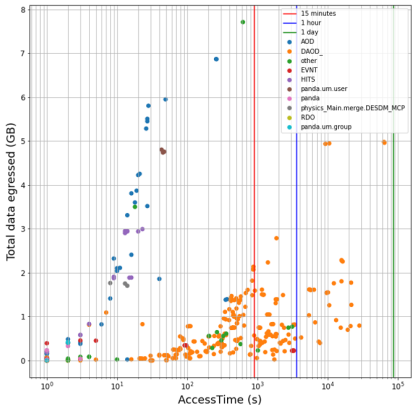

Figure 3. This chart shows the total data transferred at fileClose as reported by XRootD monitoring messages, grouped by file type. After analysing data transfers through the XRootD-PFC we found a large amount of transfers occured within a 1s window and were assigned to have been transferred within 0s. Datapoints in Blue correspond to AOD files within ATLAS workflows and Datapoints in Orange correspond to DAOD files within ATLAS workflows.

4.2 Throughput Results

Figure 4 shows the total data egressed from the XRootD-PFC service as collected by the 3 different parsers in section 3.3. XRootD only reports on the total amount of data that is read when a file is finally closed. As this data was collected from an XRootD-PFC instance used by worker nodes it was found that the granularity for determining how much data has been broadcast has to therefore be very large. This is caused by a common workflow to open a file for access and then to slowly stream data from it as needed over many hours.

When comparing the data collected by the different parsers we find a good agreement between them. This is powerful as it allows us to cross-validate the data collected from XRootD monitoring messages.

Unfortunately we also find a strong disagreement with the throughput rates measured by XRootD-PFC and the same egress data metric collected by the node_exporter service.

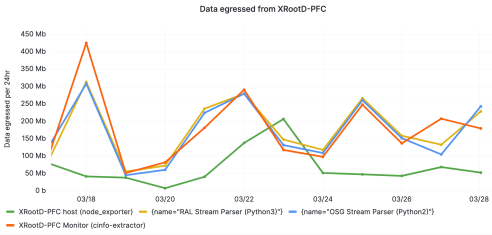

Figure 4. This chart shows the total data egress from the XRootD-PFC as reported by XRootD monitoring messages (red, blue, yellow) and the total data egress from the XRootD-PFC as reported by node_exporter (green).

4.3 Isolated Testing Results

After a disagreement was found between egress metrics collected from XRootD monitoring messages and node_exporter, a 2nd test setup was deployed to verify the correctness of the data collected from XRootD monitoring messages. This simpler test setup was only interested in extracting metadata metrics from the .cinfo files. This has shown to be representative of other internal metrics due to the strong agreemnt between the 3 parsers in section 3.3.

Figure 5 shows the results of this test. When a file is accessed using a normal read method we find that the data reported by XRootD monitoring messages is consistent with the data reported by node_exporter.

However, when using the xrdcp copy tool within XRootD we find that the data reported by XRootD monitoring messages is inconsistent the filesize on disk. Unfortunately due to the limited testing here we can't identify if this is a problem within XRootD or if this shows that extra data is being copied over the wire than is required.

It is worth emphasising that although Figure 5 shows a disagreement, the actual final file that was copied to a local filesystem was *always* identical to the original file in size and checksum.

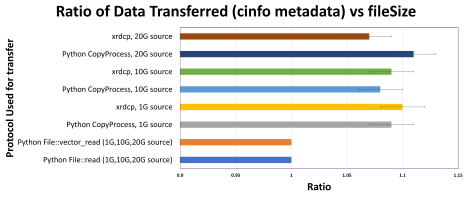

Figure 5. This chart shows the ratios of data egressed from the XRootD-PFC as reported by XRootD monitoring messages vs the filesize on disk.

5 Discussion

As shown in section 4.2 care needs to be taken when analysing data collected from XRootD monitoring messages. Figure 1 demonstrates one of the problems when trying to understand the collected metrics from XRootD-PFC. This is that metrics relating to throughput are only broadcast on a file being closed. When examining the broadcast metrics from XRootD this

can make the amount of data egressed appear extremely bursty. When correctly re-analyzing the similar data as shown in Figure 4 we find the data egressed seems more sensible.

With this in mind we find that trying to use XRootD metrics to uderstand the throughput of a service is challenging. In the case of larger sites involved in data migrations this might be less of a concern due to the short time that a transfer would be expected to take. For the use-case of a Tier2 XRootD-PFC, XRootD metrics are potentially only useful for examining past behaviour. Realtime collected metrics will always lag by the length of time that many files are open which can be as high as 48hr.

Further comparing the results of these metrics to the results collected from node_exporter we find that there is a strong disagreement between them. This suggests a problem within XRootD's handling of the metadata itself. This is potentially emphasised by the lack of reproducibility in reported metrics when copying the same file multiple times.

All of the original studies were performed with XRootD version v5.4.0, however further testing has revealed the problems identified in isolated testing still remain as of XRootD v5.6.2 the latest version of XRootD at the time of writing.

6 Conclusions

After trial and error, we believe we've understood the problems that have previously made interporating the reported metrics from XRootD monitoring messages challenging. This has allowed us to use these metrics to potentially understand the performance of the XRootD-PFC service at the Edinburgh Tier2 site.

As we look to primarily support ATLAS workflows at Edinburgh we have found that a simple optimization for an XRootD-PFC moving forward would be to prioritize the caching of DAOD files over AOD. This could potentially achieved by simply not caching AOD file accesses which would result in a reduction in the amount of data that needs to be written to the cache. This would likely translate into an improved performance of a XRootD-PFC in production. We are keen to pursure further testing of this in the future.

In addition to testing file-access and throughput, isolated testing has shown that the monitoring metrics reported by XRootD are actually non-reproducible.

With this in mind we recommend extreme caution with any attempt to use the XRootD monitoring metrics for constructing reliable throughput measurements as they don't appear to reflect the actual behaviour of the service.

References

[1] A. Hanushevsky, simonmichal, L. Janyst, G. GANIS, F. Furano, A.M. Tadel, B.P. Bockelman, P. Elmer, J.L. Salmon, J. Becla et al., *xrootd/xrootd: v5.6.2* (2023), `https://doi.org/10.5281/zenodo.8348959`

[2] Currie, Robert, Yuan, Wenlong, EPJ Web Conf. **251**, 02052 (2021)

[3] *Node exporter*, `https://github.com/prometheus/node_exporter`

[4] *Prometheus*, `https://github.com/prometheus/prometheus`

[5] D. Weitzel, B.P. Bockelman, jthiltges, M. Zvada, M. Selmeci, *opensciencegrid/xrootd-monitoring-collector: Initial tagged release* (2021), `https://doi.org/10.5281/zenodo.4670589`

[6] A. Lahiff, *Xrootd parser for ral xrootd-pfc instances*, `https://github.com/stfc/xrootd-mon-to-Elasticsearch`

[7] *Watchdog*, `https://github.com/gorakhargosh/watchdog`