EPJ Web of Conferences **295**, 01056 (2024) https://doi.org/10.1051/epjconf/202429501056
CHEP 2023

XRootD Client: a robust technology for LHC Run-3 and beyond

Guilherme Amadio[1,*], *Cedric* Caffy[1], *Andrew Bohdan* Hanushevsky[2], *Michał Kamil* Simon[1], and *David* Smith[1]

[1]CERN
[2]SLAC

Abstract. During the LHC era the XRootD framework has proven to be a critical component of numerous data management and software defined storage solutions (most importantly EOS, the CERN storage technology used for the LHC experiments), and as such grew into one of the most strategic storage technologies in the High Energy Physics (HEP) community. Over the last year significant developments in the area of the XRootD client have been introduced, making it even more reliable and robust, as well as easier to debug. Here, we present an overview of the new XRootD client and its main features, namely, support for erasure coding, in-flight data integrity checks, and the new record plug-in and replay tool that allow to record an I/O pattern and then replay it for debugging or benchmarking purposes.

1 Introduction

The challenges for Run-3 of the LHC [1], which began in 2022, include managing the massive amounts of data generated by the LHC experiments and ensuring their efficient storage, access, and analysis. In order to address these challenges, XRootD [2, 3] is now widely used by the High Energy Physics (HEP) community. XRootD was initially developed to meet the data storage and access requirements of the BaBar experiment [4–6] at SLAC and later extended to meet the needs of experiments at the Large Hadron Collider (LHC) [7, 8]. XRootD supports multiple data access models, including hierarchical, file-based, and object-based access, and it can operate on various types of storage systems, including disk, tape, and cloud storage. It is designed for scalable, high-performance, and fault-tolerant data access, transfer, and management.

XRootD has become a critical component in the data management strategy of the LHC along its history, which spans more than 20 years. In the early days, it was developed as a replacement for `rootd` within ROOT [9, 10], therefore its name. In the early 2000s, the code was moved into its own repository, and XRootD became an independent project. It was briefly rebranded as Scalla in the early 2010s [8], but the old name proved too popular, so it was later renamed back to XRootD. At the time of this writing, XRootD is in its fifth major version, which introduced support for encryption with TLS and other major features discussed in the next sections.

*e-mail: amadio@cern.ch

2 The XRootD Client

Early in XRootD's development history, the client was part of the ROOT repository [10]. The first production version of the standalone XRootD client [11–13], *XrdClient*, was added to the repository in Sep 2004 by Fabrizio Furano. This version of the client introduced important features such as support for parallel asynchronous requests for data block reads, coalescing of requests to avoid multiple requests for the same data, and a memory cache to store the coalesced requests. In the second half of 2012, Lukasz Janyst introduced a newly rewritten, thread-safe version of the client, *XrdCl*, into the repository. This is the version still in use today. Its debut happened in XRootD 4.0, at which point *XrdClient* was declared obsolete. In XRootD 5.0, the code for *XrdClient* was finally removed from the repository.

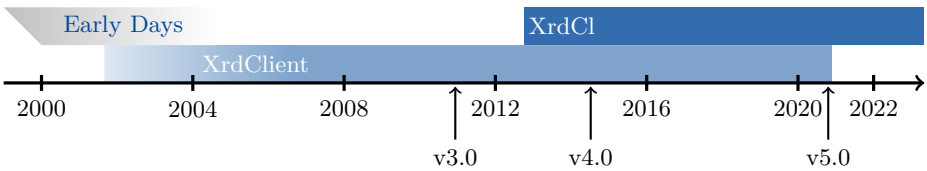

Figure 1: XRootD client development timeline.

The XRootD client is the main tool used to transfer experimental data into and out of the data center at CERN. In 2022, over 570 PB of data have been written into EOS physics instances by the main LHC experiments. The XRootD protocol accounted for the majority of this amount, as it can be seen in the statistics data shown below.

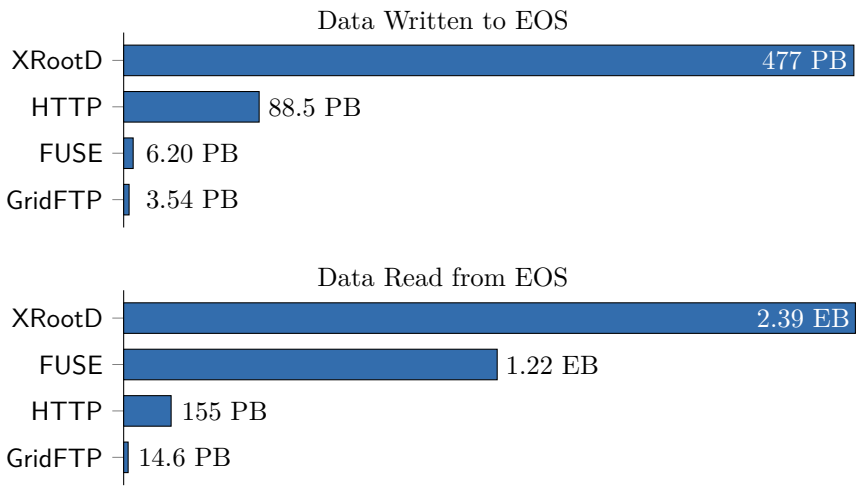

Figure 2: Data written and read on EOS physics instances in 2022.

Whilst the volume of data written is already massive, it is dwarfed by the amount of data read for analysis. In the same year of 2022, about 3.78 EB of data have been

read from the data center, 2.4 EB of which using the XRootD protocol, either by the standalone XRootD client, `xrdcp`, or via XRootD client code integrated into other applications, such as ROOT. Moreover, since the EOS FUSE client also relies on the XRootD client code in its implementation, it means that reads via FUSE are also served by XRootD.

3 Erasure Coding Plugin (XrdEc) v5.2

Running the LHC is expensive, hence the possibility of data loss has to be minimized. Traditionally, data durability has been achieved by replicating data at CERN as well as distributing copies across the world within the Worldwide LHC Computing Grid (WLCG). This strategy, however, may become prohibitively expensive at the increased data rates to be produced by the High-Luminosity LHC (HL-LHC). The solution, implemented originally for EOS and later in XRootD, is to use erasure coding to provide redundancy at much smaller overhead than simple data replication. Erasure coding works by dividing data into chunks and adding parity blocks that can be used to reconstruct the original data in the event of hardware failure.

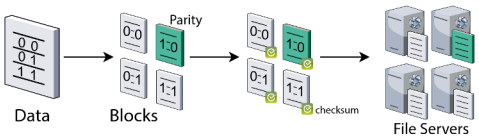

Figure 3: Erasure Coding in an Nutshell.

When configured to use erasure coding, the server will automatically split files into n data blocks and k parity blocks according to the configuration and distribute each piece to a different disk or file storage server as depicted below.

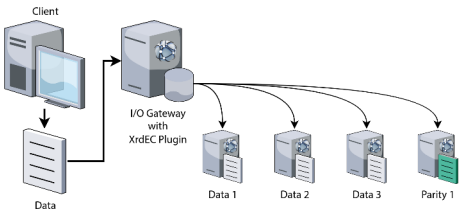

Figure 4: Writing Data via I/O Gateway with Erasure Coding Plugin.

When reading, the data can be reconstructed either at the server, in case the client does not have support for erasure coding enabled, or each block can be read directly if the client has support for erasure coding enabled. The client can also write data directly in this case.

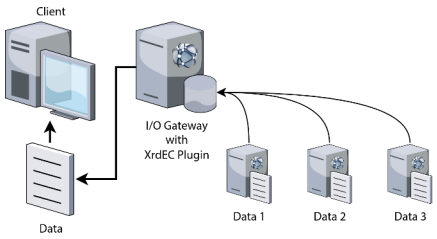

Figure 5: Reading Data via I/O Gateway with Erasure Coding Plugin

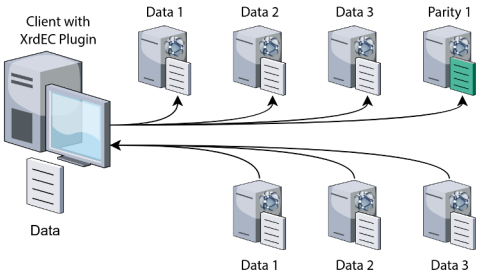

Figure 6: Client with Erasure Coding Plugin can Read/Write Data Directly

4 Declarative Asynchronous API v5.0–v5.4

The declarative API for the XRootD client [14, 15] has been created with erasure coding support as its main use case. Its main objective is to simplify composition of a series of asynchronous operations on one or more remote files, such as writing a data block striped into n data chunks and k parity chunks in parallel. Its main advantage is that it allows composing multiple asynchronous operations together without requiring error-prone boiler plate code in the process, as shown in the code listing below.

```
using namespace XrdCl;
void ECWrite(uint64_t offset, uint64_t size, const void *buffer,
             ResponseHandler *handler) {
  /* calculate number of chunks */
  std::vector<Pipeline> writes(nchunks);
  for (size_t i = 0; i < nchunks; ++i) {
    /* calculate offset, size, and buffer for each chunk */
    File f = new XrdCl::File();
    Pipeline p = Open(file, url, flags)
      | Parallel(  Write(file, chunk_offset, chunk_size, chunk_buffer),
                   SetXAttr(file, "xrdec.cksum", checksum) )
      | Close(file) >> [file](XRootDStatus&) { delete file; };
  }
  Async(Parallel(writes) >> [handler](XRootDStatus&) {
    handler->HandleResponse(new XRootDStatus(), 0);
  });
}
```

Listing 1: XRootD client declarative API example.

5 Data Integrity (pgRead/pgWrite) v5.0–v5.5

Disk failures can cause corruption for data at rest, but transmission errors while the network is under heavy load may also lead to data corruption across the wire. For small file transfers, this sort of corruption is not a problem, since the cost of retransmitting the data is low in case a checksum mismatch occurs at the end of the transfer on the destination. However, when transferring large files (>10GB), a better mechanism is necessary. Therefore, in addition to data durability features like erasure coding for data at rest, XRootD has also introduced pgRead and pgWrite to ensure data integrity for data tranfers across the network. With pgRead/pgWrite, data is checksummed at 4K page boundaries, and if any transmission errors occur, only pages with mismatched checksums need to be retransmitted, greatly improving reliablility.

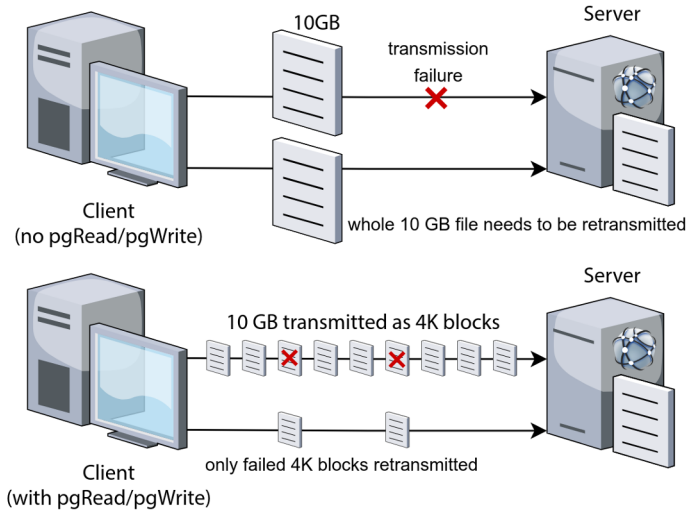

Figure 7: XRootD data integrity checks reduce cost of data retransmission.

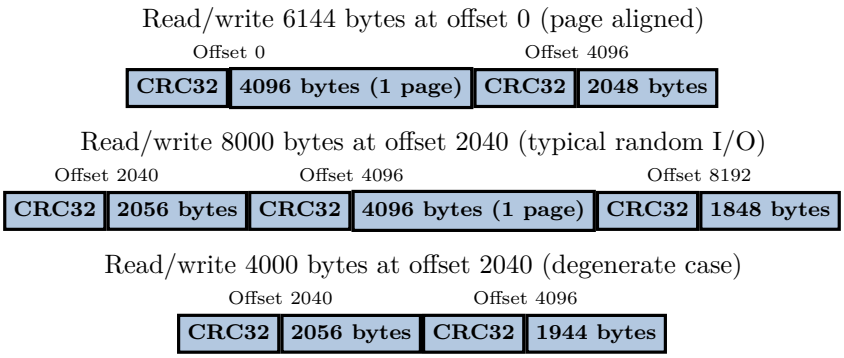

Figure 8: Wire Layout for Data Transfers with pgRead/pgWrite.

6 Record/Replay Plugin v5.5

The recorder plugin for the XRootD client, introduced in XRootD 5.5.0, allows users to record remote data access patterns in a way that is transparent to client applications. It can be enabled by the user by creating a configuration file in its home directory at `$HOME/.xrootd/client.plugins.d/recorder.conf` with the following contents:

```
url = *
lib = /usr/lib64/libXrdClRecorder-5.so
enable = true
output = /tmp/xrdrecord.csv
```

This configuration will instruct the XRootD client to load the recorder plugin and record each operation into the `/tmp/xrdrecord.csv` file on disk. A simple example is shown below using ROOT to run an RDataFrame tutorial that reads CMS opendata via XRootD.

```
$ root.exe -l -b -q df102_NanoAODDimuonAnalysis.C

Processing df102_NanoAODDimuonAnalysis.C...
Info in <TCanvas::Print>: pdf file dimuon_spectrum.pdf has been created
Events with exactly two muons: pass=31104343 all=61540413 eff=50.54
Muons with opposite charge: pass=24067843 all=31104343 eff=77.38
```

The output file can be inspected and replayed with the `xrdreplay` command line tool. Without any options, it will read the CSV file and execute again the same operations, taking care of reproducing the timings from the original run as well. When run with the `-p` option, it produces a summary of the operations performed by the client:

```
$ xrdreplay -p /tmp/xrdrecord.csv
# ================================================
# IO Summary (print mode)
# ================================================
# Sampled Runtime  : 23.195940 s
# Playback Speed   : 1.00
# IO Volume (R)    : 2.24 GB [ std:581.69 KB vec:2.24 GB page:0 B ]
# IO Volume (W)    : 0 B [ std:0 B vec:0 B page:0 B ]
# IOPS      (R)    : 147 [ std:72 vec:75 page:0 ]
# IOPS      (W)    : 0 [ std:0 vec:0 page:0 ]
# Files     (R)    : 18
# Files     (W)    : 0
# Datasize  (R)    : 40.40 GB
# Datasize  (W)    : 0 B
# ------------------------------------------------
# Quality Estimation
# ------------------------------------------------
# Synchronicity(R) : 100.00%
# Synchronicity(W) : 0.00%
```

In the summary above, we can see that ROOT read 2.24 GB of data in 23.2 s from the remote file, mostly using vector reads.

7 Conclusion and Future Work

We have presented the new XRootD client, *XrdCl*, and its main features. As we approach the high luminosity phase of the LHC, XRootD is evolving to cope with the higher volume of data and to offer better performance and observability for remote data analysis and storage. In the future we plan to extend these features further and use them to optimize XRootD, EOS, and remote analysis applications using ROOT.

References

[1] M. Arsuaga-Rios, V. Bahyl, M. Batalha, C. Caffy, E. Cano, N. Capitoni, C. Contescu, M. Davis, D.F. Alvarez, J. Guenther et al., *LHC Data Storage: Preparing for the Challenges of Run-3*, in *25th International Conference on Computing in High-Energy and Nuclear Physics* (2021), Vol. 251, p. 02023, `https://doi.org/10.1051/epjconf/202125102023`

[2] *xrootd/xrootd: v5.6.1* (2023), `https://doi.org/10.5281/zenodo.8135874`

[3] A. Hanushevsky, A. Dorigo, F. Furano, *The next generation root file server*, in *14th International Conference on Computing in High-Energy and Nuclear Physics* (2005), pp. 680–683, `https://cds.cern.ch/record/865679/files/p680.pdf`

[4] B. Aubert et al. (BaBar), Nucl. Instrum. Meth. A **479**, 1 (2002), `hep-ex/0105044`

[5] A. Adesanya et al. (BaBar Computing Group), *On the Verge of one petabyte: The Story behind the BABAR database system*, in *13th International Conference on Computing in High-Energy and Nuclear Physics* (2003), Vol. C0303241, p. MOKT010, `cs/0306020`

[6] A. Hanushevsky, A. Trunov, L. Cottrell, *Peer to peer computing for secure high performance data copying*, in *12th International Conference on Computing in High-Energy and Nuclear Physics* (2001)

[7] S. Campana, D.C. van der Ster, A. Di Girolamo, A.J. Peters, D. Dullmann, M. Coelho dos Santos, J. Iven, T. Bell, *Commissioning of a CERN production and analysis facility based on xrootd*, in *18th International Conference on Computing in High-Energy and Nuclear Physics*, edited by S.C. Lin (2011), Vol. 331, p. 072006

[8] A. Hanushevsky, D.L. Wang, *Scalla: Structured Cluster Architecture for Low Latency Access*, in *26th IEEE International Parallel and Distributed Processing Symposium* (2012)

[9] *ROOT Data Analysis Framework*, `https://root.cern`

[10] A. Dorigo, F. Furano, P. Elmer, A. Hanushevsky, *XTNetFile, a fault tolerant extension of ROOT TNetFile*, in *14th International Conference on Computing in High-Energy and Nuclear Physics* (2005), pp. 996–999, `https://cds.cern.ch/record/865754`

[11] A. Hanushevsky, *Hyper-scaling data access: understanding XROOTD data clusters*, in *ROOT Workshop 2005* (2005)

[12] F. Furano, A. Hanushevsky, Journal of Physics: Conference Series **119**, 072016 (2008)

[13] F. Furano, A. Hanushevsky, J. Phys. Conf. Ser. **219**, 072005 (2010)

[14] R. Poenaru, M. Simon, *C++ Declarative API – Implementation Overview Within the XRootD Framework*, in *19th RoEduNet Conference: Networking in Education and Research* (2020)

[15] M. Simon, A. Hanushevsky, EPJ Web Conf. **251**, 02063 (2021)