Physics Data Forge: Unveiling the Power of I/O Systems in CERNs Test Infrastructure

Guilherme Amadio[1,*], *Luca* Mascetti[1], *Andreas Joachim* Peters[1], *Andrea* Sciabà[1], and *David* Smith[1]

[1]CERN

Abstract. Efficient remote file access is essential for High Energy Physics (HEP) experiments, particularly with the anticipated tenfold increase in data volume for Run 4 of the LHC. XRootD and EOS are critical components of the HEP ecosystem, enabling remote data access via XRootD and HTTP protocols. In this work, we evaluate the performance of various client implementations over these protocols using an XRootD server with a high-performance `tmpfs` mount. In order to ensure an infrastructure-independent assessment, benchmarking is conducted in a controlled 100GbE networking environment, eliminating external bottlenecks and isolating software performance. Network integrity was first validated using `iperf3` before executing the benchmarks. Our study identifies a performance bottleneck in multi-stream data transfers at high rates within the XRootD client. A fix has been developed and will be included in XRootD 5.8.0, significantly improving high-throughput data access.

1 Introduction

XRootD [1–3] is a scalable, high-performance data access system developed to address the challenges of distributed storage in High Energy Physics (HEP). Originally designed at SLAC [4], it has become a cornerstone of CERN's computing infrastructure [5, 6], enabling efficient remote access to petabyte-scale datasets. XRootD provides a modular architecture, its own native XRootD protocol for data access as well as HTTP, and is integral to EOS [7, 8], CERN's flagship storage solution.

With the upcoming High Luminosity Large Hadron Collider (HL-LHC) [9], data rates are expected to increase by an order of magnitude. This surge necessitates optimizations in data access strategies, including enhanced protocol implementations, improved client-server interactions, and advancements in storage and networking infrastructure.

In this study, we evaluate the performance of XRootD and HTTP clients, focusing on benchmarking throughput, identifying bottlenecks, and implementing performance optimizations. To ensure accurate results, all tests are conducted in a controlled high-performance networking environment, minimizing external interference and isolating software-specific performance characteristics.

*e-mail: amadio@cern.ch

https://doi.org/10.1051/epjconf/202533701339

2 Test Environment and Hardware Configuration

The benchmarking tests were conducted on two high-performance computing nodes at CERN, each designed to sustain the high-throughput demands of a 100GbE networking environment.

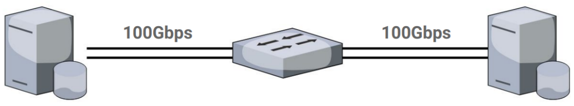


Figure 1: CERN test environment: two compute nodes connected to each other with 100GbE networking via a single network switch.

Both nodes are equipped with dual AMD EPYC 7302 16-core processors, a Mellanox ConnectX-5 network adapter supporting 100Gbps Ethernet connectivity, and 256GB of RAM. For storage, each node features two 2TB NVMe SSDs. It is important to note that the network adapters are attached to the first socket in each compute node, while the storage devices are attached to the second socket, as shown in figure 2 below. This means that to serve data from the storage devices, the data needs to pass through the interconnect between the two sockets after being read before being sent out to the network, which negatively affects performance.

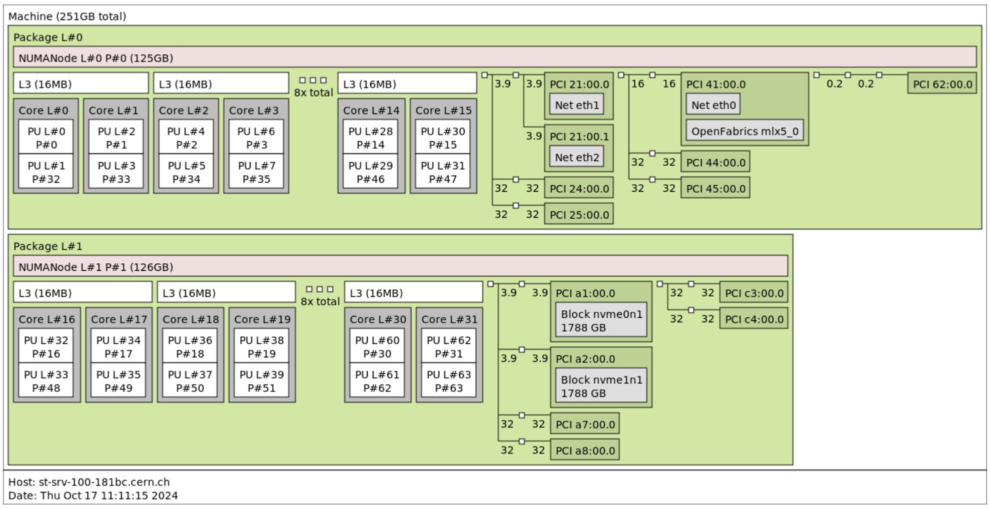


Figure 2: Node topology. Each node is dual socket, with networking devices attached to the first socket, and storage devices attached to the second socket.

The software stack included Alma Linux 8.10, running Linux kernel 4.18.0-553, alongside XRootD 5.7.1 [3] (latest available version at the time the work was carried out) and the following HTTP client implementations: Davix 0.8.7 [10, 11], curl 7.61.1 [12], and wget 1.19.5 [13]. Some benchmarks also feature a comparison with scp, from OpenSSH 8.0p1 [14].

The server node was configured with a 128GB tmpfs mount to serve files directly from memory. The client node was used to download 10GB and 100GB files from the server node under various scenarios.

3 Network Configuration and Optimization

In order to minimize network-related bottlenecks, we applied some standard optimizations for 100GbE networks to our testing environment. First, however, we validate the speed of the network using `iperf3`:

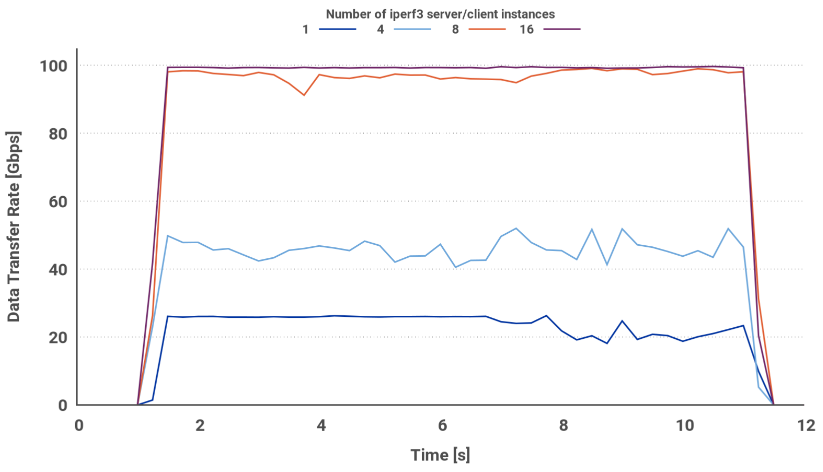


Figure 3: Network speed verification with iperf3 (MTU=1500).

Although we can achieve the maximum speed of the link using 16 `iperf3` processes, we note that for 1, 4 and 8 processes the performance is not as stable as it could be. Network adapters are connected only to the first socket, so we use a fixed number of 8 `iperf3` processes to verify that stability improves by pinning each process to a physical core on the first socket (light blue line in figure 4 below). In subsequent tests, processes are pinned in the best configuration, that is, one process per physical core.

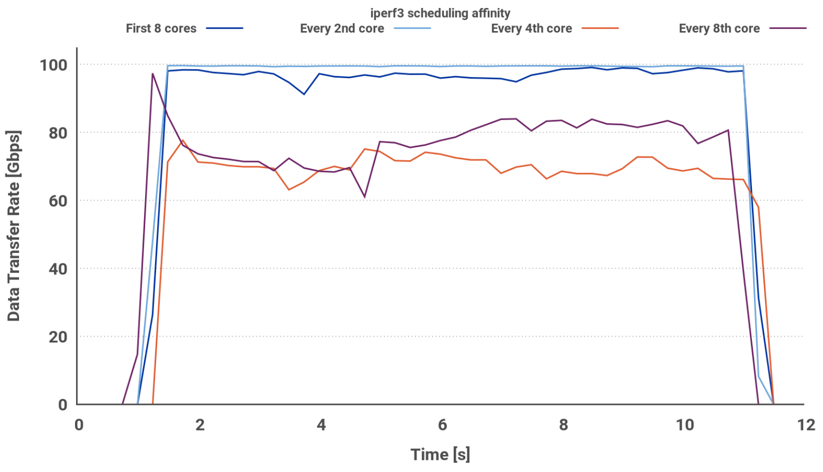


Figure 4: Network speed with iperf3 with different scheduling affinity.

Next, we increase the MTU (Maximum Transmission Unit) to 9000 bytes (Jumbo Frames) to reduce packet overhead and improve efficiency for large data transfers.

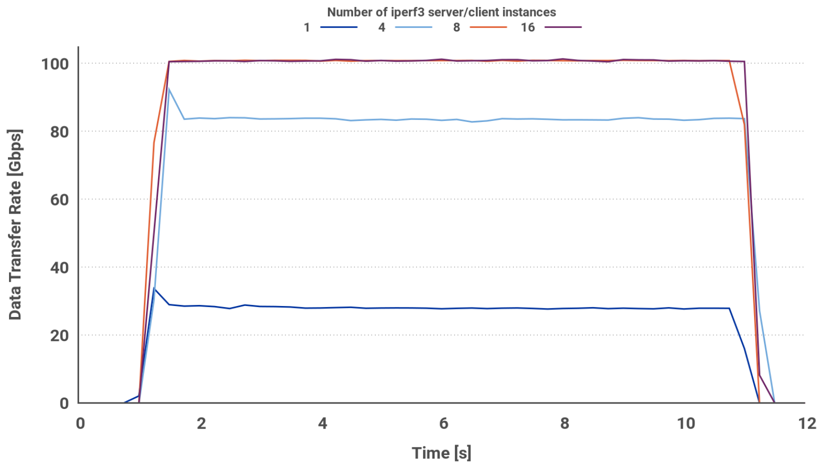


Figure 5: Network speed verification with iperf3 (MTU=9000).

Finally, we optimize other parameters of the network. TCP congestion control was set to BBR, an advanced algorithm designed to optimize throughput over high-bandwidth links by dynamically adjusting congestion windows. TCP buffer sizes were increased, ensuring that large amounts of in-flight data could be efficiently managed without hitting system-imposed limits. The list below shows the actual settings applied:

```
net.ipv4.tcp_wmem = 4096 65536 2147483647
net.ipv4.tcp_rmem = 4096 87380 2147483647
net.core.rmem_max = 2147483647
net.core.wmem_max = 2147483647
net.core.default_qdisc = fq
net.ipv4.tcp_congestion_control = bbr
net.ipv4.tcp_mtu_probing = 1
net.core.optmem_max = 1048576
```

Additionally, NIC ring buffers were expanded to their maximum allowable size to prevent packet drops and improve performance during high-throughput operations.

```
$ ethtool -g eth0
Ring parameters for eth0:
Pre-set maximums:
RX:     8192
TX:     8192
Current hardware settings:
RX:     8192
TX:     8192
```

These optimizations helped create an ideal testing environment, allowing precise evaluation of different client implementations without interference from underlying network bottlenecks.

4 XRootD and HTTP Client Benchmarks

To assess the performance of different remote data access clients, we conducted a series of benchmarks. Each benchmark involves downloading the same files containing randomly generated data, and writing the output into `/dev/null` in order to avoid performance bottlenecks from storage devices. No authentication mechanisms were enabled, since we are only interested in pure data transfer performance. However, some tests use TLS encryption to evaluate its impact on transfer speeds. XRootD also supports a mode with TLS encryption on the control stream, while data streams remain unencrypted. We added this mode to our tests for completeness. Figures 6 and 7 below show the download speed and download time for a single-stream transfer of a 10GB and a 100GB file, respectively. Each measurement is the average of 3 consecutive runs of the client.

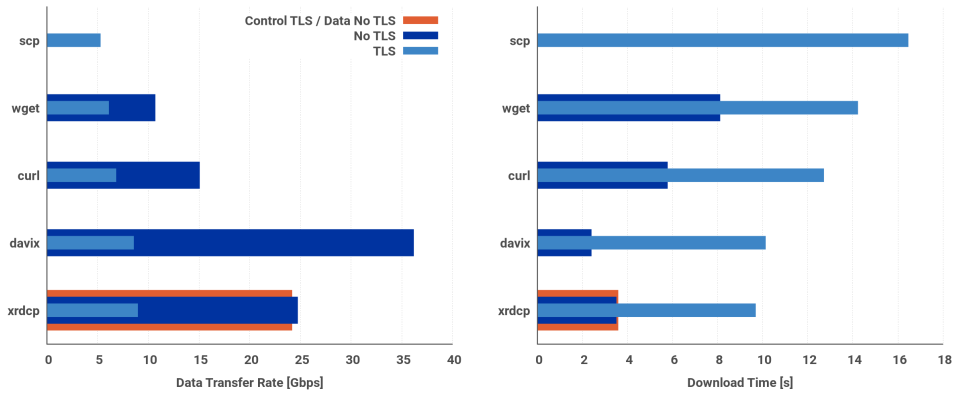


Figure 6: Download speed and download time for 10GB file.

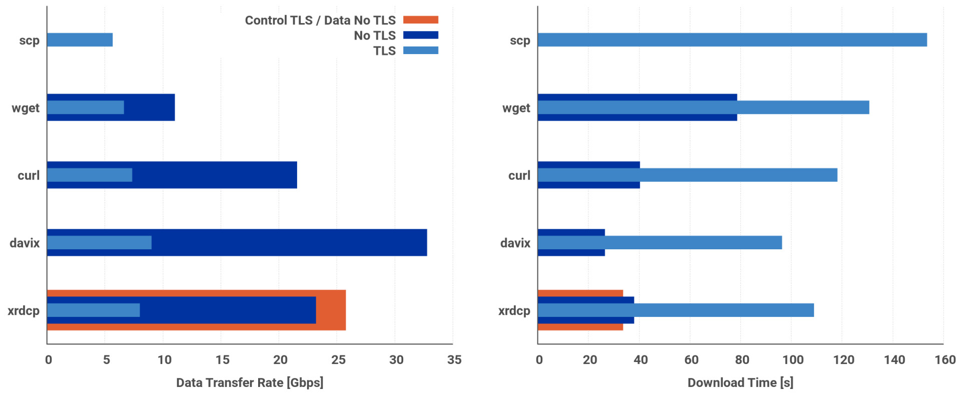
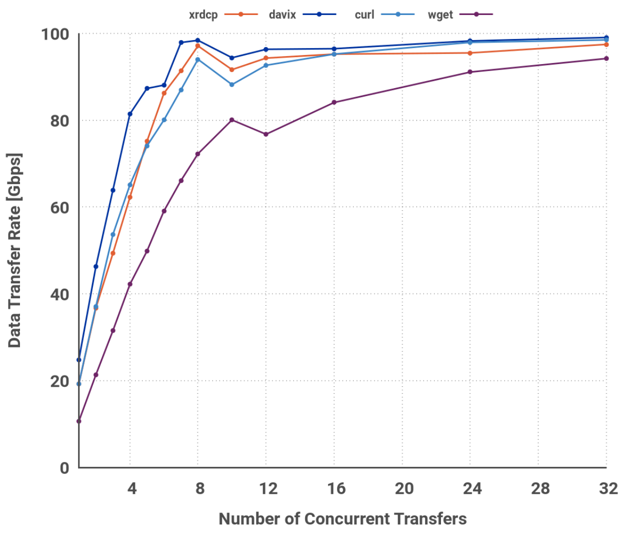


Figure 7: Download speed and download time for 100GB file.

Davix is the best performing client, while XRootD comes in second with similar performance to curl. The performance difference between XRootD and Davix can be explained at least in part by the fact that HTTP clients only have to issue a single

`GET` request to the server to get the file streamed back to them, while the XRootD client normally divides the file to be downloaded into chunks and issues a read request to the server for each chunk.

Since single-stream transfers are not suffient to saturate the speed of the network link, we achieve saturation by running concurrent transfers. Figure 8 shows the aggregated transfer speeds for up to 32 concurrent transfers with and without TLS encryption enabled.



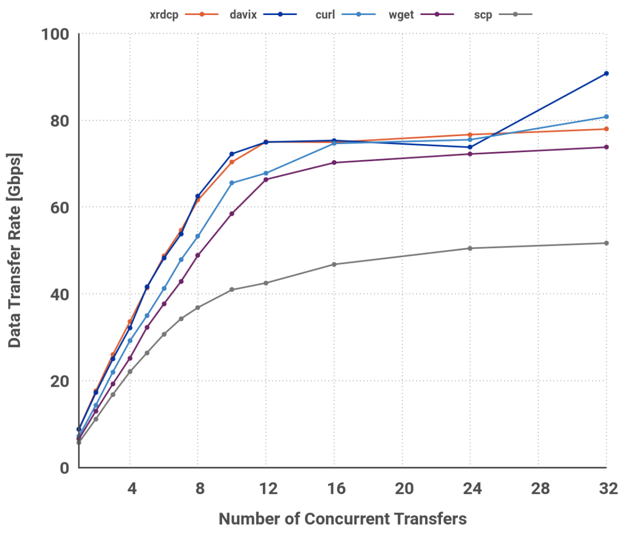


Figure 8: Concurrent transfers without (top) and with (bottom) TLS encryption.

With the exception of `wget`, all other clients can reach near saturation at 8 concurrent transfers when TLS encryption is not enabled. Due to its high CPU cost, transfers with TLS enabled reach a plateau of about 75Gbps after 12 concurrent transfers.

Another way to achieve saturation, or so we thought, would be to use the multi-stream feature of the XRootD client. We measured transfer rates varying the number of data streams up to 16 to analyze its effect on throughput. To our surprise, on this optimized networking setup, adding streams did not significantly change performance: Upon analyzing the client performance, we could see that on high-speed networks, the

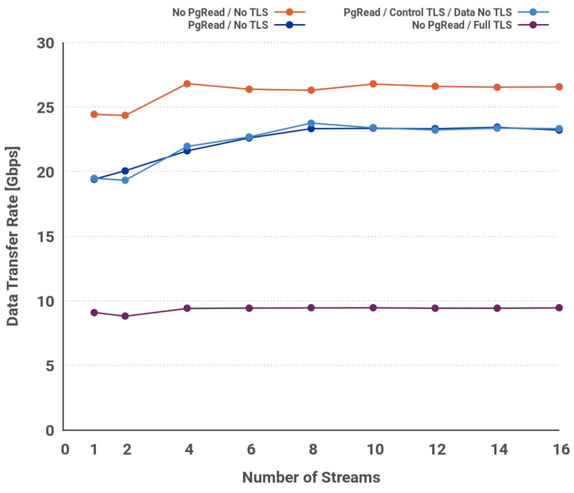


Figure 9: Multi-stream download performance of XRootD 5.7.1 client.

client quickly becomes CPU-bound. This is shown in figure 10 below. There is one thread that is always busy (shown continuously in blue in the inset), while other threads remain mostly idle (scattered blue boxes show their sparse activity). The

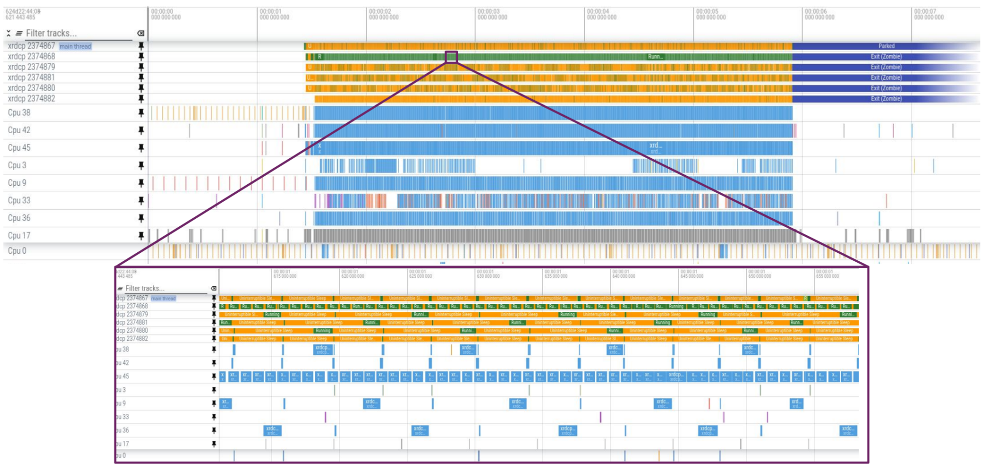


Figure 10: XRootD client in 100GbE networking environment. One thread is CPU bound, while other threads remain mostly idle.

reason for the client becoming CPU-bound was well understood, however. For each connection (i.e. channel), additional data streams are created by opening additional sockets. However, these sockets were all mapped to use the same polling thread, which is the thread that becomes CPU bound in the end. A fix for this issue involved remapping sockets to pollers based not on their channel, but on their file descriptors instead, as shown on the right in figure 11. After this performance bug was fixed, we

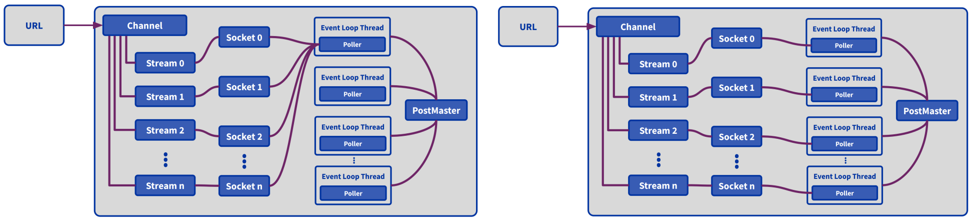


Figure 11: XRootD client architecture before (left) and after (right) the performance bugfix.

obtained the expected results when using multiple streams for data transfers, shown in figure 12 below. Now using 8 streams is sufficient to saturate the link with a single multi-stream transfer when TLS encryption is not enabled.

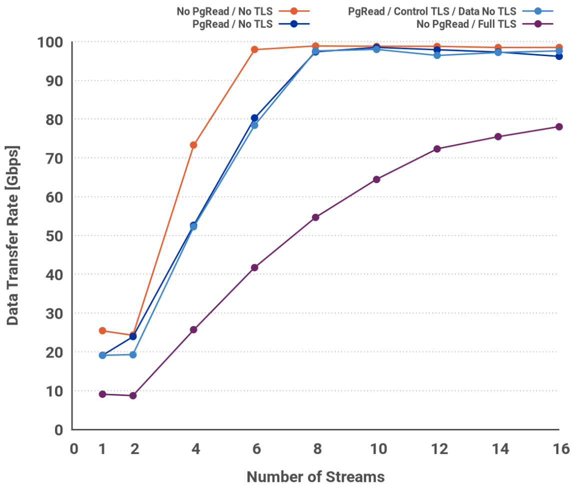


Figure 12: Multi-stream download performance before (left) and after (right) performance bugfix in XRootD client.

5 Conclusion

Our study demonstrates that while XRootD can sustain high-performance data access for HL-LHC workloads, optimizations in networking, multi-threading, and client implementations are still necessary as we move to high-performance networking environments. Future work will explore further performance testing on high-speed networks beyond 100Gbps speeds and in realistic situations, as recently done in [15, 16], simulating packet loss and latency, as well as adding support for disk and network `io_uring` [17] to XRootD and to achieve throughputs beyond 200Gbps for HL-LHC.

References

[1] 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`

[2] *XRootD Project*, `https://xrootd.org`

[3] *XRootD 5.7.1* (2024), `https://doi.org/10.5281/zenodo.13682245`

[4] 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)

[5] R. Gardner, S. Campana, G. Duckeck, J. Elmsheuser, A. Hanushevsky, F.G. Hönig, J. Iven, F. Legger, I. Vukotic, W. Yang (ATLAS), *Data federation strategies for ATLAS using XRootD*, in *J. Phys. Conf. Ser.*, edited by D.L. Groep, D. Bonacorsi (2014), Vol. 513, p. 042049

[6] 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

[7] A. Peters, L. Janyst, *Exabyte Scale Storage at CERN*, in *International Conference on Computing in High Energy and Nuclear Physics (CHEP 2010)* (2011), Vol. 331, p. 052015, `https://doi.org/10.1088/1742-6596/331/5/052015`

[8] *EOS Open Storage*, `https://cern.ch/eos`

[9] G. Apollinari, I.B. Alonso, O. Brüning, T. Nakamoto, L. Rossi, *High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report V.0*, Number CERN-2017-007-M in CERN Yellow Reports: Monographs (2017), `https://dx.doi.org/10.23731/CYRM-2017-004`

[10] A. Devresse, F. Furano, *Efficient HTTP Based I/O on Very Large Datasets for High Performance Computing with the Libdavix Library*, in *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, edited by J. Zhan, R. Han, C. Weng (Springer International Publishing, 2014), pp. 194–205, ISBN 978-3-319-13021-7

[11] F. Furano, A. Devresse, O. Keeble, M. Hellmich, A. . Ayllón, *Towards an HTTP Ecosystem for HEP Data Access*, in *Journal of Physics: Conference Series* (2014), Vol. 513, p. 032034, `https://dx.doi.org/10.1088/1742-6596/513/3/032034`

[12] D. Stenberg, *curl and libcurl*, `https://curl.se`

[13] *GNU Wget*, `https://www.gnu.org/software/wget/`

[14] *OpenSSH 8.0p1*, `https://www.openssh.com/releasenotes.html#8.0p1`

[15] Fajardo, Edgar, Arora, Aashay, Davila, Diego, Gao, Richard, Würthwein, Frank, Bockelman, Brian, *Systematic benchmarking of HTTPS third party copy on 100Gbps links using XRootD*, in *25th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2021)* (2021), Vol. 251, p. 02001, `https://doi.org/10.1051/epjconf/202125102001`

[16] A. Arora1, J. Guiang, D. Davila, F. Würthwein, J. Balcas, H. Newman, *400Gbps benchmark of XRootD HTTP-TPC*, in *26th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2023)* (2024), Vol. 295, p. 01001, `https://doi.org/10.1051/epjconf/202429501001`

[17] J. Axboe, *Efficient IO with `io_uring`*, `https://kernel.dk/io_uring.pdf`