EPJ Web of Conferences **245**, 04034 (2020)
CHEP 2019
FERMILAB-CONF-20-786-SCD
https://doi.org/10.1051/epjconf/202024504034

XRootD Third Party Copy for the WLCG and HL-LHC

T Adye[1]*, B Bockelman*[2]*, K Ellis*[1]*, O Freyermuth*[3]*, F Furano*[4]*, G Ganis*[4]*, A Hanushevsky*[5]*, H Ito*[6]*, I Johnson*[1]*, O Keeble*[4]*, D Litvintsev*[7]*, A Manzi*[4]*, P Millar*[8]*, T Mkrtchyan*[8]*, G Patargias*[1]*, A Rossi*[7]*, H Severini*[9]*, M Simon*[4]*, E Sindrilaru*[4] *and W Yang*[5,*]on behalf of the WLCG TPC Working Group

[1]Science and Technology Facilities Council, United Kingdom
[2]Morgridge Institute for Research, United States
[3]University of Bonn, Germany
[4]European Organization for Nuclear Research (CERN), Switzerland
[5]SLAC National Accelerator Laboratory, United States
[6]Brookhaven National Laboratory, United States
[7]Fermi National Accelerator Laboratory, United States
[8]Deutsches Elektronen-Synchrotron (DESY), Germany
[9]University of Oklahoma, United States

Abstract. A Third Party Copy (TPC) mechanism has existed in the pure XRootD storage environment for many years. However, using the XRootD TPC in the WLCG environment presents additional challenges due to the diversity of the storage systems involved such as EOS, dCache, DPM and ECHO, requiring that we carefully navigate the unique constraints imposed by these storage systems and their site-specific environments through customized configuration and software development. To support multi-tenant setups seen at many WLCG sites, X509 based authentication and authorization in XRootD was significantly improved to meet both security and functionality requirements. This paper presents architecture of the pull based TPC with optional X509 credential delegation, and how it is implemented in native XRootD and dCache. The paper discusses technical requirements, challenges, design choices and implementation details in the WLCG storage systems, as well as in FTS/gfal2. It also outlines XRootD's plan to support newer TPC and security models such as token based authorization.

1 Motivation

A key task of the distributed computing model adapted by the Large Hardon Collider (LHC) experiments [1] is to move the data around the world efficiently. Sitting at a third location, an automated data processing engine or data management system of an experiment (the client) can trigger data file transfers from one of its data depots (a.k.a. sites) to another. This is referred to as Third Party Copy (TPC). During a TPC operation, the client maintains control channel connections with both data sites but data are transferred from one site to the other without going through this client at the 3rd location.

During the last two decades, the LHC experiments uses GridFTP [2] to fulfil the task of TPC. With the change of the supporting model and business model of GridFTP, and the growing maturity of other remote data access protocols currently used by the LHC experiments, the Worldwide LHC Computing Grid (WLCG) [3] has set up a working group

[*] Corresponding author : yangw@slac.stanford.edu

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

to explore using alternative mechanisms to fulfil the TPC requirement. In specific, TPC mechanisms that exist in the HTTP protocol and XRootD protocol [4] are chosen for further development and evaluation. This paper covers the development of TPC based on the XRootD protocol to meet the requirement from the WLCG community.

2 The scope of the work

To replace GridFTP for the task of TPC for LHC experiments, the XRootD TPC needs to be able to do a number of things:

1. work with the existing data transfer management software FTS [5] and gfal2 [6], and optionally be supported by Rucio [7].
2. fit into the existing Grid computing's security mode that uses GSI and VOMS for authentication and authorization [8].
3. work with all WLCG storage systems. This includes EOS, dCache, DPM, Storm, XRootD, CEPH and Posix file systems (The Storm storage systems are covered as Posix file systems).
4. scale up its capability to meet the need of High Luminosity LHC (HL-LHC) [9].
5. accommodate the newer JSON Web Token (JWT) [10] based authentication and authorization model defined by the WLCG for the HL-LHC computing.

In this paper, we will discuss work related the items 1, 2 and 3, as well as current stress testing related to item 4. Together, items 1-4 consist of the phase 1 work of the WLCG TPC Working Group. Item 5 is a newly added task as a potentially future work (phase 2 work). It will not be covered in this paper.

The XRootD protocol and XRootD service are originally implemented in C++ and used by all of the above WLCG storage systems except dCache. The dCache storage system uses a Java implementation for its XRootD door. This paper will cover the XRootD TPC related work under both implementations.

3 The mechanism of the XRootD TPC

XRootD implemented a TPC protocol for Third Party Copy [11], using the rendezvous copy paradigm. To adapt the existing WLCG X509/VOMS authentication and authorization model, a subset of the TPC protocol (TPC Lite) is constructed. Both C++ and Java implementations support the TPC protocol.

3.1 XRootD TPC protocol with rendezvous paradigm

The TPC protocol in XRootD uses a rendezvous copy paradigm revolving around a client-supplied rendezvous key. In essence, the client generates a rendezvous key and supplies it to the destination when it initiates a TPC process. The client then waits for the destination to initiate a child process to pull the data from the source. Once this new child process starts, the client (or trigger (TRG) in Fig.1) establishes a rendezvous point at the source. The destination is given a limited amount of time to rendezvous with the source using this rendezvous key. After these steps are completed successfully, the destination pulls the file from the source.

The rendezvous copy paradigm itself is a way for the destination to authenticate with the source. It does not require additional authentication and authorization between the source and destination, and thus is independent of the usual security requirement for upload and download. Though several measures were implemented in the XRootD TPC protocol itself to prevent the rendezvous key from being stolen and misused, security sensitive use cases should consider using this mechanism along with the XRootD protocol over SSL, which completely encrypts all traffic, including the exchange of the rendezvous key.

3.2 XRootD TPC Lite protocol for credential delegation

At the time when we started this work, we did not have a capability to run XRootD protocol over an encrypted SSL connection. But since GSI and VOMS infrastructure is widely used in the WLCG and LHC experiments, we adopted a revised approach: when the client is authenticated with the destination via GSI, the destination requests the client to delegate its X509 proxy (plus the VOMS attributes) to the destination server. This delegated X509 proxy will later be used by the child TPC process (as mentioned above) to authenticate with the source and then fetch the data.

The process of client credential delegation happens within the XRootD GSI security plugin, after the client and the destination server's GSI plugin establish symmetric encryption key, and before the TPC procedure starts. A number of steps and improvements were implemented to safeguard the transportation of delegated credentials. With the credential delegation, the rendezvous key exchange is no needed. This revised TPC protocol is a subset of the original TPC, and thus named TPC Lite.

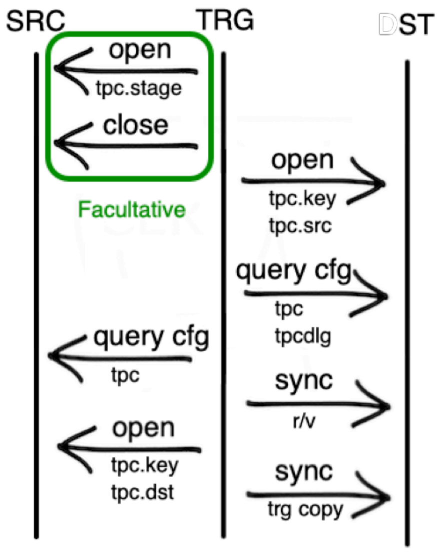

Figure 1. Event sequence in the XRootD TPC protocol. TRG is the client that triggers the TPC. The CGI key tpc.dlg will be set to zero to indicate that there will be no credential delegation. The last two steps between SRC and TRG establish a rendezvous point. The tpc.key is used to pass the rendezvous key.

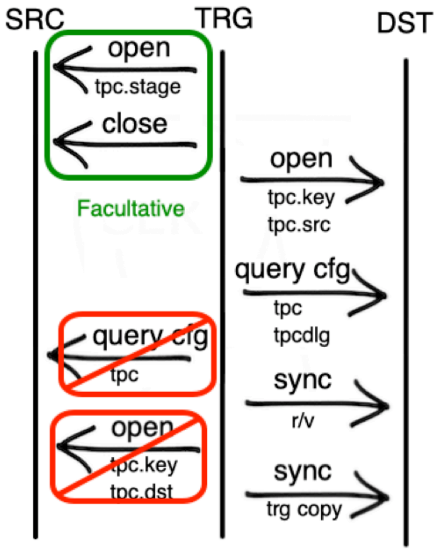

Figure 2. Event sequence in the XRootD TPC Lite protocol. The steps to establish rendezvous point (in Fig. 1) is removed. The CGI key dpc.dlg=1 indicates that client credential will be delegated (also an indication that the destination server should perform according to the TPC Lite protocol). The tpc.key will be empty. Note: the credential delegation happens outside of the TPC Lite protocol.

4 The challenges in implementation

In the phase 1, we addressed a number of challenges to meet our goal of developing a secure, reliable, high performance and scalable XRootD TPC Lite. The following describes some of the major work we completed.

4.1 Coordinated development of C++ and Java implementation

As mentioned before, the XRootD TPC Lite protocol is a newly defined subset of the original XRootD TPC protocol. Both C++ and Java implementations followed the same specification. Coordinated work, complex or simple, happened in almost every step during the TPC Lite implementation. We will not go into details but this mention is just to acknowledge the effort to make the two implementations work seamlessly.

4.2 Safeguard the Diffie-Hellman parameter exchange

When an XRootD client and an XRootD server negotiate and shake hand using the XRootD GSI security plugin, they exchange their Diffie-Hellman (DH) parameters [12] in order to establish a symmetric encryption key for additional data exchange. Until recently, in the C++ implementation the DH parameters from the server to the client was not signed by the private key of the server. It was possible for a mid-man between the client and the server to intercept the DH parameter exchange process and insert its own DH parameters separately with the client and the server. By doing so, the mid-man could decrypt the message passing by. This security vulnerability hadn't been a problem because there was no use case of data exchange after a symmetric encryption key was setup. X509 proxy delegation changed this.

When implementing X509 proxy delegation, we wanted to use the symmetric key to encrypt the exchange of delegated proxy. We therefore enhanced the DH parameter exchange procedure by asking the server (in both C++ and Java implementations) to sign the DH parameters it sends to the client. This enhancement effectively prevents mid-men. Without this safeguard, a mid-man can steal the delegate credential from the client.

4.3 Enforce RFC 2818

The HTTP over TLS standard (RFC 2818) [13] specifies that a host's canonical name (CNAME) and all DNS aliases should be included in the host certificate's CNAME and Subject Alternative Name fields. This prevents someone who controls the DNS service from sending the client to a rogue server. We implemented RFC 2818 in the XRootD GSI security plugin to prevent the client from unknowingly delegating its credential to a roger server. The client is responsible to enforce this RFC, and should explicitly agree on delegating its X509 credential. If the RFC can not be enforced, the GSI authentication process will continue but the client should refuse to delegate its X509 credential.

4.4 Eliminate memory leaks

As a long running service, XRootD TPC service can tolerate very little memory leak. In the C++ implementation, we did find significant memory leak in both XRootD GSI security module and the VOMS attributes extraction function. Most of the problems are related to using an old API in the OpenSSL library [14]. This API works fine on the Red Hat Enterprise Linux 5 platform [15] (and equivalent). But on Red Hat Enterprise Linux 6 and later platforms with newer OpenSSL (openssl-1.0.1e or later), we have to switch to use a newer, supported API in order to avoid memory leak.

4.5 TPC request forwarding

The EOS storage system at CERN has a special topology among its front end XRootD proxy server nodes. While all these nodes participate in downloading and uploading data via the XRootD protocol, only a subset of them participate in XRootD TPC. To accommodate this topology, we implemented a feature that allows non-TPC nodes to forward TPC requests to TPC capable nodes.

4.6 Improve multiple TCP stream performance

During an XRootD TPC, the destination starts a child process to pull data from the source. For scalability reasons, it is preferred that this child process uses as little memory as possible, and utilizes as much network bandwidth as possible, and if necessary, uses multiple TCP streams to improve bandwidth utilization.

This child process can be any XRootD client. In our evaluation, we used a standalone xrdcp [16] from the XRootD release 4.11.0rc2, to copy a 512 MB RAM disk file from SLAC to CERN. To find the optimal setting and to guide further improvement, we tuned two parameters related the data chunk size and the number of asynchronous requests [15] to adjust the sizes of the non-shared rolling buffers and the number of buffers. We also combined those non-shared rolling buffers to a shared rolling buffer.

Since the tests were conducted over a wide area network (WAN), the WAN fluctuation needed to be considered. Measurements conducted in different days are not comparable due to significant fluctuation. We automated the measurements process in order to complete the measurement in a short period of time (usually a few hours). During the process, we intertwined different settings, and repeated the measurement 10 times to reduce the impact of network fluctuation. After many tries, we found that setting the above two parameters to 256 KB and 16 respectively (compare to the default values of 8MB and 4) gave both smaller memory footprint and slightly optimal performance. Table 1 shows the time (in seconds) it took to transfer the above 512 MB file in RAM disk with xrdcp, using multiple TCP streams, with either the default setting or the small footprint setting

Table 1. Performance comparison with multiple TCP streams

Setting	1 TCP stream	2 TCP stream	3 TCP stream
Default	248±40	129±25	107±22
Small footprint	227±39	127±30	92±29

We also compared xrdcp performance with that of the “bbcp” [17], which is a standalone WAN data transfer tool widely used in the HPC community. Noting that xrdcp and bbcp take different steps to establish the initial connections and additional TCP connections, we made this comparison under the solo purpose of understanding whether there was significant room to improve the performance of xrdcp. Table 2 shows the comparison of the two when copying a 512 MB file.

Table 2. xrdcp and bbcp performance are comparable

	2 TCP stream	4 TCP stream	8 TCP stream
xrdcp with small footprint	36.8	15.2	8.7
bbcp	24.2	17.0	13.6

Later releases of XRootD were optimized based on the measurements in Table 1. The xrdcp memory usage (RSS) is reduced from approximately 36MB to 16MB when using 1 TCP stream, and from approximately 79MB to 25MB when using 4 TCP streams.

Note that using multiple TCP streams will improve performance only when the XRootD server supports asynchronous I/O. The C++ implementation supports this feature.

4.7 Working with Rucio, FTS and gfal2

Several LHC experiments are moving towards using Rucio as their data management system. Through FTS and gfal2, Rucio drives the data transfer among sites. The work in Rucio, FTS and gfal2 to support XRootD TPC is not covered in this paper. At the XRootD software side, transfer marks, a kind of heartbeat information are sent from the destination to the FTS server periodically to help FTS to detect broken data transfers.

5 Deployment and tests

The current deployment of the XRootD TPC covers all WLCG storages systems. In some cases, these storage systems are accessible from the Internet. In other cases, they are behind firewalls. A solution to enable the XRootD TPC service in the latter scenarios is to setup a XRootD proxy server [18] at the site border. Functioning as a data transfer node (DTN), this proxy node is accessible from the Internet via the GSI security protocol. The DTN may access the backend storage using any site preferred security protocol. In most cases, the backend storages are configured to trust the operations initiated by the DTNs, and leave the responsibility of authentication and access control to the DTNs. Scaling up is achieved by using the XRootD's native clustering mechanism to setup a cluster of DTNs.

To test and validate the TPC development, the WLCG TPC Working Group deploys a daily smoke test and almost continuous stress tests. Many sites participate in the tests. The smoke test is a functional test. It checks the upload and download functions at all participating sites. It also uses a reference site to conduce in and out XRootD TPC against all other sites. The smoke test currently covers all types of WLCG storages; The stress tests run against site endpoints that closely resemble the production setup. Stress tests have an important role of exposing load related issues.

In additional to the XRootD TPC, the WLCG TPC Working Group is also developing the HTTP(s) protocol based TPC, using either C++ XRootD software stack or Java/dCache. The HTTP TPC is not covered by this paper. We are able to run both XRootD TPC and HTTP TPC on the same instance at SLAC endpoint.

6 Future work

There are two main areas that we will continue to work on. They are the TLS support in the XRootD protocol, and the support of JWT based authentication and authorization in XRootD. Similar to the HTTPS protocol, the XRootD protocol over TLS (the XRootDs protocol) provides transport layer encryption. In addition to encrypting data traffic, this capability allows us to use an authentication and authorization infrastructure (AAI) other than GSI and VOMS in the XRootD TPC (and not limit ourselves to the XRootD TPC Lite). Given that JWT is the main next phase task for the WLCG Working Group, developing a JWT security plugin in XRootD is an obvious priority.

References

1. The Large Hadron Collider, accessed at https://home.cern/science/accelerators/large-hadron-collider
2. W. Allcock, "GridFTP: Protocol Extensions to FTP for the Grid. Global Grid" ForumGFD-R-P.020, 2003.
3. Worldwide LHC Computing Grid (WLCG), accessed at http://wlcg.web.cern.ch
4. xrootd.org, accessed at http://www.xrootd.org
5. File Transfer Service, accessed at https://fts.web.cern.ch/.
6. Grid File Access Library, accessed at https://dmc.web.cern.ch/projects/gfal-2/home
7. RUCIO Scientific Data Management, accessed at https://rucio.github.io/
8. VOMS, accessed at https://italiangrid.github.io/voms/
9. High Luminosity LHC, accessed at https://home.cern/science/accelerators/high-luminosity-lhc
10. JSON Web Token (JWT), accessed at https://tools.ietf.org/html/rfc7519
11. Third Party Copy Protocol, accessed at https://xrootd.slac.stanford.edu/doc/dev49/tpc_protocol.htm
12. R. Merkle. "Secure Communications Over Insecure Channels". *Communications of the ACM*. **21**(4): 294–299, 1978
13. HTTP over TLS, accessed at https://tools.ietf.org/html/rfc2818
14. OpenSSL Cryptography and SSL/TLS Toolkit, accessed at https://www.openssl.org/
15. Red Hat Enterprise Linux, accessed at https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux
16. Xrdcopy, accessed at https://xrootd.slac.stanford.edu/doc/man/xrdcp.1.html
17. BBCP, accessed at https://www.slac.stanford.edu/~abh/bbcp/
18. Proxy Storage Services Configuration Reference, accessed at https://xrootd.slac.stanford.edu/doc/dev410/pss_config.htm