C++ Declarative API and Pyxrootd - An Overview Within Xrootd Framework

Robert Poenaru  
DFCTI  
NIPNE-HHMagurele, Romania  
robert.poenaru@protonmail.chMichał Simon  
Information Technology Department  
CERNGenève, Switzerland  
michal.simon@cern.ch

*Abstract*—A brief description of the Xrootd architecture and its purpose within the WLCG group, together with an overview of the server- and client- sides of the Xrootd framework are provided within the present work. The client side of Xrootd has a relatively new implementation called Declarative API. Its main purpose is to provide the user an asynchronous interface that is more in line with the modern C++ paradigm. A focus on the development workflow for this API is given. Moreover, the pyxrootd package, which provides a python interface with the Xrootd client, is also discussed and tested in a usual file-operation use case.

Keywords—Xrootd, pyxrootd, asynchronous programming, declarative API, pipeline, server, client.

# Introduction

Started as a protocol which granted remote access to root format specific files, with a primary use case focused on data analysis (rather than data transfer), Xrootd became widely used within then scientific community at CERN (European Organization for Nuclear Research) and other large-scale facilities (e.g., SLAC-Stanford Linear Accelerator Center). Over the last years, the framework evolved a lot, and it now supports data analysis, data transfers, data management plus features like staging data from tape.

In terms of storage capacity, only the ATLAS and CMS collaborations alone produce a total of around 150 Petabytes of data which needs to be accessed by thousands of physicists within the Worldwide Large Hadron Collider Compute Grid - WLCG community [7]. As a result, a key objective of the WLCG is to assure both the process of moving the data between sites and deliver the data to any end-user application. Even though LHC has proven to be able moving data at the necessary throughput [2], only by adopting a so-called federated regional storage using Xrootd will be able to avoid any potential latency issues for physicists when accessing the data and the eventual complexity of the tools involved. A discussion on the storage federation will be given in the following sections.



1. Process of accessing data in the “jobs go to data” mode. User submits an analysis job that is sent through the compute grid to the sites that host the data.

The addition of Xrootd on top of the Anydata, Anytime, Anywhere project (or AAA for short) [2] allowed the high Energy Physics community to achieve global data storage federations that have a single data-access entry point and also a common data-access protocol, which changed the old paradigm of distributed multi-tiered storage. This could be possible through the hierarchical deployment of redirectors that allow site discovery that have available data in real-time. Although, Xrootd supported multi-storage deployments for a long time, the addition of a feature which allowed its proper functionality within a global, multi-site environment was in fact the core idea of AAA.

In order to emphasize the importance of Xrootd, it is worth mentioning that currently at CERN, the main storage solution is EOS – a technology developed in-house and built on top of Xrootd framework, with some additional features. Experiments within LHC (e.g. ATLAS, CMS, LHCb, ALICE) but also smaller experiments (e.g. AMS) have EOS as a native solution for data storage/access for the users. In the following section, we provide a clearer picture of the Xrootd framework, both in terms of its server side as well as its client side, since both implementations are crucial in understanding the overall workflow of data access and data manipulation within WLCG community.

# Xrootd framework

The main objective of any scientific project that is based on experiments which ran at CERN (but also to the other places within WLCG) is the access to the compute resources which are used for submitting jobs that aim to solve a particular task. An old model of such a workflow is described in diagram below (also called “jobs go to data” paradigm [2]).

## Whenever the user wants a local copy of the data that is studied, a change of workflow must be made from using the data analysis tools to the usage of data transfer toolset (which is usually provided by the experiment’s computing facility or the grid middleware [2]).

This induces a lot of extra-time (overhead) that can lead to a relatively slow progress into tackling the actual tasks which have to be performed by the physicist with the required data. Federated storage system is the implementation that aims at solving such issues. Defined in [2] as a collection of unpaired storage resources that are managed by a set of domains which are cooperating with each other (but also independent) and also are accessible via a common namespace. By using multiple dedicated Xrootd servers at each site and a centralized redirector (an in-depth description of an Xrootd redirector can be found in [2]), it is possible to build these storage spaces where the user makes a direct contact with the central endpoint and is redirected to a site which can provide the necessary data.



1. Xrootd server architecture.

The change between the local analysis and Grid analysis is removed within a federated storage system and on top of that, data can be accessed independent on the location, which also reduces the latency between data request and data access.

## Server-side Xrootd

At its core, Xrootd acts like any remote data server, however, it does seem innovative in terms of its scalability, robustness, fault tolerance, job recovery (in case of job failing during an execution), cache mechanism and many more. In fact, a tremendous work (progress) has been done in the recent years, especially for extending the scalability features (e.g., [1], [8]), caching [2] and many more. In fact, the development team is constantly committing new or improved features, and these are documented on the official repository [10]. Other characteristics of an Xrootd server that assure a high-performance data availability are load-balancing, optimal use of hardware resources (like sockets, memory, CPU), cooperation between different (Xrootd) servers, minimize the number of jobs that have to be restarted due to a server or network problem.

The Xrootd server architecture is composed of four main layers, namely: Network layer, Protocol layer, File-system layer and a Storage layer (see Fig. 2). Being developed on a run-time plug-in mechanism, new features can be added into the framework with little effort. The main reasons for developing the Xrootd within a layered model are the optimization of specific functionalities while minimizing resource usage and isolation of services allows for dynamical loading (determining in this way which implementation works best in a particular environment [1]).

## Client-side Xrootd

The client part (XrdClient) of the Xrootd framework is built as a ROOT and POSIX compliant system [1], and its core functionality provide implementations like:

* A communication protocol: this allows for requesting access to an Xrootd server through an authentication process and also giving access to the desired data resources which.
* Handler for any communication errors. This is done through a development of high-level communication policies at the client side.
* Connections to a server need to support multiple and independent data streams.

Xrootd client is made up of three layers, each with a different characteristic [3].

### The interface layer.

### High-level communication layer.

### Low-level communication layer.

The client is taking care of data caching, pipelining, parallelizing and aggregating requests which will provide benefits in terms of latency and throughput. This layered implementation of the XrdClient object assures data accessing methods (e.g. access files, create files, manipulate the data) but also the optimization of the data accessing process - request aggregation [3]. The POSIX interface of the XrdClient is implemented through a shared library which takes any POSIX specific call and routes it to the XrdClient.

# Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections A-D below for more information on proofreading, spelling and grammar.

##### Acknowledgments

Thank you all.

##### References

1. Dorigo, A., Elmer, P., Furano, F., & Hanushevsky, A. (2005, March). XROOTD/TXNetFile: a highly scalable architecture for data access in the ROOT environment. In Proceedings of the 4th WSEAS International Conference on Telecommunications and Informatics (p. 46). World Scientific and Engineering Academy and Society (WSEAS).
2. Bauerdick, L., Benjamin, D., Bloom, K., Bockelman, B., Bradley, D., Dasu, S., ... & Lesny, D. (2012, December). Using xrootd to federate regional storage. In Journal of Physics: Conference Series (Vol. 396, No. 4, p. 042009). IOP Publishing.
3. Boeheim, C., Hanushevsky, A., Leith, D., Melen, R., Mount, R., Pulliam, T., & Weeks, B. (2006). Scalla: Scalable cluster architecture for low latency access using xrootd and olbd servers. Technical report, Stanford Linear Accelerator Center.
4. Fajardo, E., Tadel, A., Tadel, M., Steer, B., Martin, T., & Würthwein, F. (2018, September). A federated Xrootd cache. In Journal of Physics: Conference Series (Vol. 1085, No. 3, p. 032025). IOP Publishing.
5. Gardner, R., Campana, S., Duckeck, G., Elmsheuser, J., Hanushevsky, A., Hönig, F. G., ... & Yang, W. (2014, June). Data federation strategies for ATLAS using XRootD. In Journal of Physics: Conference Series (Vol. 513, No. 4, p. 042049).
6. Simon, M. (2019, March 08). XRootD Client Configuration & API Reference. Retrieved November 03, 2020, from https://xrootd.slac.stanford.edu/doc/xrdcl-docs/www/xrdcldocs.html
7. The Worldwide LHC Computing Grid (WLCG), http://wlcg.web.cern.ch/
8. De Witt, S., & Lahiff, A. (2014). Quantifying XRootD scalability and overheads. In Journal of Physics: Conference Series (Vol. 513, No. 3, p. 032025). IOP Publishing.
9. Pyxrootd: Python bindings for XRootD. Retrieved November 03, 2020, from https://xrootd.slac.stanford.edu/doc/python/xrootd-python-0.1.0/
10. Xrootd: The central GitHub repository, [Source Code available on November 03, 2020]: <https://github.com/xrootd/xrootd>.