8 September 2023
agalavottib@gmail.com

Improving XRootD's testing suite

Angelo Galavotti
IT-SD-PDS Summer Student 2023
Supervised by: Guilherme Amadio

Summary

This document presents a comprehensive exploration of the enhancements made to the testing methodologies within the XRootD project. We explore the transition from legacy testing frameworks like CppUnit to modern alternatives, particularly GoogleTest, as well as the introduction of a local test data generation and the optimization of testing processes by replacing a container-based approach with one based on local instances of XRootD.

The result is not only a significant increase in code coverage but also a reduction in testing time. Tests now run seamlessly within the CI pipeline with every commit. We conclude by addressing future opportunities, emphasizing the potential for further code coverage expansion through the additions of tests related to authorization methods and Zip archives.

Contents

1 Introduction

XRootD[1] is a data management software designed to provide high performance, scalable and fault tolerant access to various types of data repositories. Originally conceived through collaboration between CERN and Stanford University's SLAC National Accelerator Laboratory, it has since gained widespread adoption within the High Energy Physics (HEP) community.

It 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.

Initially, the software was built in order to tackle the challenges for Run-3 of the LHC, which began in 2022. These include managing the massive amounts of data generated by the experiments and ensuring their efficient storage, access, and analysis.

Figure 1: The XRootD logo

2 Overview

2.1 XRootD's testing suite

XRootD's testing suite plays a pivotal role in ensuring the reliability and functionality of the software. It offers a comprehensive framework for systematically evaluating various aspects of XRootD's components and features, including its data access, transfer, and management capabilities.

The suite covers a wide range of use cases, including using redirector hosts for accessing information in data servers, as well as multithreaded file transfer operations. By validating the software's behavior under different conditions, it contributes to its stability and robustness.

To execute these comprehensive tests seamlessly, the suite leverages CMake's testing component *ctest*, which significantly streamlines the integration of tests in the CI as well as facilitating their configuration.

2.1.1 Previously employed structure and technologies

In the context of my assignment during my time as a Summer Student, there are certain aspects of the original suite that hold the most significance:

- XRootD employs *CppUnit* as the main library of its testing suite. CppUnit is a widely used C++ unit testing framework designed for the creation of unit tests for C++ codebases.

[1]The repository of the project can be found at https://github.com/xrootd/xrootd

As a port of the famous JUnit library, it allows to define test cases and assertions in a structured way.

- In order to rigorously assess certain functionalities related to server clusters, XRootD utilizes a series of interconnected Docker containers in a hierarchical structure, as to replicate real-world distributed environments used in *High Energy Physics* (HEP) experiments.

  Such configurations often involve redirector servers, referred to as *managers* and *meta-managers*, along with data servers, which store the actual data requested by users.

- In addition, pre-generated files containing random data of substantial size are employed in order to evaluate its file transfer, read, and write capabilities.

2.1.2 Flaws

The importance of the aforementioned testing methodologies comes from the fact that they possess inherent shortcomings. In particular:

- **CppUnit**. Although once widely utilized, CppUnit is now considered somewhat obsolete due to a lack of consistent maintenance and updates. This is also backed by the fact that the latest stable release dates back to 2017[2], raising concerns about its compatibility with modern development practices.

  As software development continues to advance, relying on a testing framework with limited ongoing support may not be the best choice, and could render development tasks more tedious. More contemporary alternatives are required in order to ensure the functionality of the tests and, consequently, of the features.

- **Using Docker containers**. As previously mentioned, a significant portion of the testing suite relies on a complex Docker infrastructure, which is impractical to configure within the CI pipeline.

  The CI environment would struggle with the heavy demands of setting up the Docker image with every commit, while also complicating the collection of comprehensive test coverage data. Therefore, a more efficient and practical solution is needed.

- **Large generated test files**. The large files containing random data are downloaded from CERN's servers each time the Docker infrastructure is set up. This practice leads to substantial wastage of storage space as well as CERN's bandwidth resources.

  Addressing this issue by optimizing the method of acquiring test data can contribute to more sustainable and efficient testing procedures.

2.2 My task

Considering the context of the project, my main objective was to address said issues. To achieve this, these were some of the implemented solutions:

[2]As of time of writing of this document

- Converting tests from CppUnit to GoogleTest[3].
- Replacing the Docker-based infrastructure with one based on inter-process communication among local XRootD instances.
- Incorporating the local generation of files containing random data in the CI.

In order to perform such endeavors, a substantial amount of refactoring was invested in the process. The details regarding the implementation of such methods will be explained in the following chapters.

3 Implementation

3.1 Converting tests to GoogleTest

As previously acknowledged, the limitations of CppUnit require for a transition to a new testing framework. In this context, GoogleTest suits the criteria of this project.

GoogleTest, often abbreviated as *gtest*, is a widely embraced C++ testing framework that facilitates the creation and execution of unit tests with precision and ease. Developed by Google, this open-source tool is designed to provide a robust platform for developers to test their C++ code through assertions and test fixtures.

One notable similarity between GoogleTest and CppUnit lies in the structure of macros for the assertions . This familiarity eases the transition for developers accustomed to CppUnit's testing methodology while providing an improved and more contemporary testing framework.

CppUnit code

```
1 void SocketTest::TransferTest()
2 {
3   ...
4   CPPUNIT_ASSERT( serv.Start() );
5 ...
6 }
```

GoogleTest equivalent

```
1 TEST(SocketTest, TransferTest)
2 {
3   ...
4   EXPECT_TRUE( serv.Start() );
5   ...
6 }
```

All test cases and classes were refactored in their GoogleTest equivalent, rendering the testing suite essentially CppUnit free. Certain auxiliary libraries and headers that originally featured custom macros using CppUnit functions have undergone a transformation to their GoogleTest counterparts as well, all while retaining their original functionalities. This is evident in *GTestXrdHelpers.hh*, which houses custom procedures capable of extracting and presenting the status code for specific XRootD procedures.

Switching to GoogleTest allowed for the mitigation of some bugs, with one notable instance being the resolution of a "segmentation fault" issue encountered during multiple runs of the erasure coding tests. Such specific error ceased to reoccur following the transition.

[3]The official GoogleTest repository can be found at https://github.com/google/googletest

3.2 Replacing the Docker-based infrastructure

The testing suite adopts a cluster of communicating containers in order to simulate the real-life conditions encountered in HEP experiments. A representation of the network structure used in the tests can be found in Figure 2. In such cluster, the *metaman* and *man* servers

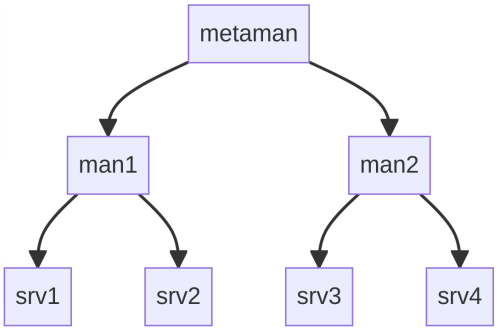

Figure 2: A scheme of the cluster structure

act as the redirector servers, while the *srv* hosts acts as data servers. The formers route the connection to servers that hosts the requested file and as such they do not contain any valuable data, whereas the latter may or may not contain the data sought after by the user.

As mentioned previously, this infrastructure was achieved through the use of Docker containers. Transitioning away from this setup to local XRootD instances running as processes necessitated some code refactoring. Specifically, it entailed removing hardcoded values referring to the servers' hostname and some replacing other features related to the previous infrastructure. Additionally, the configuration files for each host had to be rewritten accordingly. The setup and the cleanup processes of the whole cluster are integrated into CMake, and they run before and after each test.

Using local instances of XRootD instead of containers not only simplifies test execution but also enables the seamless integration of tests into the CI pipeline, allowing to gather previously unrecorded information about the code coverage of server-related features.

3.3 Generating random data locally

Previously, the testing process relied on files containing random data in order to test procedures such as reading, writing, downloading, and uploading. These files were obtained from CERN's data servers and consisted of approximately 10 files, each with a size of one gigabyte.

In the revamped approach, new, smaller files are generated locally and dynamically each time the tests are executed, and the file generation process is seamlessly integrated into the Continuous Integration pipeline. However, this transition necessitated a significant amount of refactoring. Much of the code referring to the generated files contained hardcoded values related to aspects like file size, content, or checksums. To accommodate the file changes and enhance code robustness, file information is obtained by dynamically retrieving it from these files during runtime. As such, the code remains adaptable to file variations.

3.4 Running tests in parallel

Test execution can be a time-consuming endeavor, especially when test cases are run sequentially. In an effort to substantially reduce testing time, the testing suite was modified to enable parallel test execution. This optimization involved the division of the testing suite executable into multiple executables, each associated with a specific class within the test suite. Consequently, tests that necessitated sequential execution could still proceed in that manner, while others could concurrently run in parallel.

Furthermore, each networking operation was configured to operate on different ports, ensuring smooth and conflict-free test execution.

3.5 Miscellaneous work

3.5.1 ZipTests

Although not the primary focus of my assignment, I was also tasked with the general responsibility of incorporating additional tests cases. Most of these newly created tests were centered around evaluating the Zip archive functionalities within the XrdClient software, which is a component of the broader XRootD project. These tests can be found in the *ZipTests* class.

3.5.2 smoke-test-clustered

This particular test serves as a clustered iteration of the original *smoke-test.sh* test. Like its predecessor, it involves the generation of files that are subsequently uploaded to a server and then downloaded later. However, the *smoke-test-clustered* differs in that it generates specific files, each of them unique to its respective data server. Afterwards, they are uploaded, and then downloaded back from multiple servers within a clustered environment, involving communication with a redirector.

3.5.3 Bug fixes and improving code quality

While developing tests, I managed to address certain bugs related to the XrdClient. One of these issues caused files to be be added consistently when the user was trying to open a file inside of a Zip archive, even when they did not specify such behaviour.

In addition, I implemented some general improvements aimed at the code quality, including the reduction of duplicated code and enhancements to formatting.

4 Conclusion

4.1 Results

The transformation of the testing processes has yielded significant improvements on multiple fronts. Particularly:

- The test coverage has increased from 9.4%[4] to 20.2%[5] . A considerable amount of previously uncovered procedures and files is now executed by the tests.
- Testing time is reduced of about ∼100 seconds, leading to a faster continous integration pipeline.
- The code is more robust to changes.

Equally significant is the integration of previously "ignored" tests into the CI pipeline, where they now run seamlessly with every commit thanks to the changes made. This is also part of the reason why the testing coverage has doubled, since before the tests that required a clustered infrastructure were simply not executed.

These collective enhancements expand the agility of the testing suite, and ease the delivery of more efficient, stable, and thoroughly tested software; setting a robust foundation for the ongoing evolution of the project and its successful deployment.

4.2 Future work

Looking ahead, there lie promising opportunities for the continued enhancement of the testing processes. The direction for refinement mostly revolves around increasing the code coverage, which in turn means expanding the suite by adding more tests cases. One viable approach entails the inclusion of additional test cases focused on the intricacies of Zip archives. These new tests could encompass scenarios involving file additions to the archive and structural modifications within it. Another pivotal area deserving attention pertains to authentication methods, such as VOMS Proxies, SciTokens, Macaroons, and Kerberos tickets. Given their integral role in the XRootD software, comprehensive tests related to them are of outmost importance. Additionally, broadening the versatility of the code of unit tests stands as a valuable improvement. This involves facilitating the setup of larger clusters comprising diverse hosts as well as file of different sizes, offering users greater flexibility.

Collectively, these enhancements promise to fortify the software's stability and robustness, ensuring its continued reliability as a dependable solution in the evolving software landscape.

4.3 Acknowledgements

I want to express my deep appreciation to Guilherme Amadio, my supervisor, for his support and guidance throughout the entirety of my stay at CERN, even during the most tedious moments. His mentorship has not only been instrumental in completing this task but has also equipped me with skills that I will carry with me for a lifetime. Furthermore, I extend my gratitude to Luca Mascetti, head of the IT-SD-PDS section, and to all the remarkable and insightful colleagues that I had the privilege of encountering within the IT-SD group and the broader IT department.

[4] A detailed report can be found at https://xrootd.web.cern.ch/xrootd/coverage/

[5] A detailed report can be found at https://xrootd.web.cern.ch/xrootd/coverage_new/

Lastly, my sincere thanks go to CERN and the Summer Student Team for orchestrating this incredible programme. It provided me with the opportunity to meet some of the most exceptional people in my life, an experience I will forever cherish.