



XRootD

XRootD Status and Plans

G. Amadio

EOS Workshop 2024

15 Mar 2024

XRootD Development Team (2023/2024)

- ▶ Server / OFS / OSS
 - Andrew B. Hanushevsky
 - David Smith
- ▶ Client / XrdEc / Python
 - David Smith
 - Guilherme Amadio
- ▶ CMake / Packaging / Testing / CI
 - Guilherme Amadio
- ▶ TLS
 - Andrew B. Hanushevsky
 - David Smith
 - Guilherme Amadio
- ▶ XCache
 - Alja Mrak Tadel
 - Matevž Tadel
- ▶ HTTP Protocol Plugin
 - Cedric Caffy
- ▶ HTTP Third-Party Copy (TPC) Plugin
 - Brian Bockelmann
 - Cedric Caffy
 - Elvin A. Sindrilaru
- ▶ GSI Authentication Plugin
 - Gerardo Ganis
 - Guilherme Amadio
- ▶ XrdOssCsi
 - David Smith
- ▶ SciTokens Plugin
 - Brian Bockelmann
 - Derek Weitzel
- ▶ EPEL / Debian Packaging
 - Mattias Ellert

XRootD 5.6.0 Highlights

▶ XRootD 5.6.0

- Server
 - Make **maxfd** configurable (default is 256k)
 - Use **SHA-256** by default for signatures and message digests
 - Switch to a fixed set of DH parameters (compatibility with OpenSSL 1.0.2)
 - Allow specification of minimum and maximum creation modes
 - Better detection of private IPV6 addresses (check also for **unique local address**)
 - Include token information in the monitoring stream (subject, user, vorg, role, groups)
- XCache
 - New function for file eviction
 - Allow origin to be a locally mounted directory (e.g. XCache for Ceph/Lustre)
- Client
 - New subcommand for **xrdfs cache** to allow for cache evictions
 - Do not enforce TLS when **--notlsok** option is used in combination with **root://** URL
 - Increase default number of parallel event loops to 10 (affects XCache)

XRootD/Xcache in the context of Analysis Facilities

I/O performance studies of analysis workloads on production and dedicated resources at CERN

A. Sciabà, J. Blomer, P. Canal, D. Duellmann, E. Guiraud, A. Naumann, V.E. Padulano, B. PanzerSteindel, A.J. Peters, M. Schulz, D. Smith

Analysis Grand Challenge ttbar analysis

- **Simplified analysis from CMS used as technical demonstrator in IRIS-HEP**

- Input dataset 3.6 TB, 2300 ROOT files, 1.5 GB/file consisting of CMS 2015 Open Data

- **Columnar analysis paradigm**

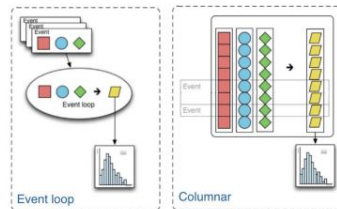
- Distributed using a map-reduce concept

- **Original Coffea implementation**

- ROOT-less, parallelism via Python futures or Dask

- **RDataFrame port (talk)**

- ROOT-based, parallelism via implicit multithreading, Dask and other



- **Measure performance and scalability**

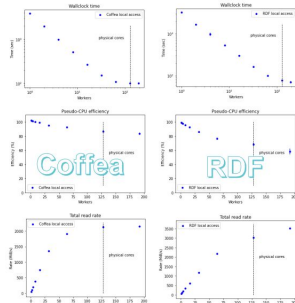
- **Local parallelism** on client node
- Data read from **local node** vs. **directly from EOS** via xrootd vs. via an **XCache** instance
- NOT a comparison between Coffea and RDF
 - Simply, different workloads with different behaviors



XRootD/Xcache in the context of Analysis Facilities

Local access

- **Scalability is excellent**
 - Some bottleneck appears for high numbers of workers
- **Overcommitting does not help for Coffea, but it increases throughput for RDF**
 - Indication that Coffea is more CPU-constrained
- **The CPU efficiency comparably high**
 - I/O not a strong bottleneck
- **Local, fast SSD storage is always going to work well**
 - Aggregate read rates up to 3 GB/s



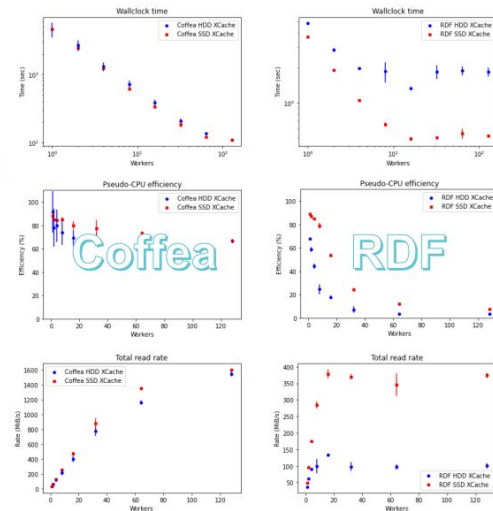
HDD/SDD-based XCache

- **Compared performance of direct access to Nebraska and CERN, cold cache and warm cache**

Coffea + HDD XCache: wallclock time (s)				RDF MT + HDD XCache: wallclock time (s)			
Site	Direct	Cold	Warm	Site	Direct	Cold	Warm
Nebraska	442 ± 16	608	133 ± 6	Nebraska	5470 ± 910	18841	1531 ± 78
EOSCMS	139 ± 8	325	137 ± 3	EOSCMS	323 ± 95	8149	1558 ± 400

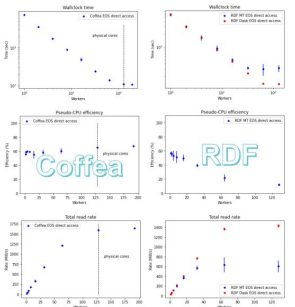
- **Performance results**

- A cold cache is slower than direct access!
- Due to sparse file access and network latency
- **Multiprocess scales very well**
- HDD XCache almost as good as SSD XCache
- RDF **multithreaded** scales very poorly "out of the box"
 - **All connections multiplexed into one ⇒ bottleneck!**
- SDD XCache helps a lot, but scalability is still broken



Direct access to EOS

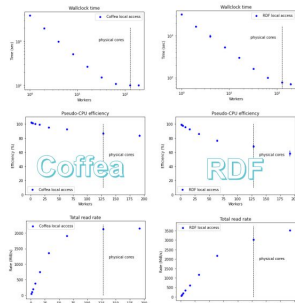
- **Scalability still good when parallelism is via multiprocess**
 - RDF implicit multithreading does not perform well with xrootd and many threads
 - RDF via Dask is multiprocess, scales better
- **CPU efficiency practically constant around 60% with multiprocess parallelism**
 - I/O time is not negligible anymore but no bottlenecks
- **Two EOS instances tested**
 - EOSCMS by default, but CERNBOX produces similar (slightly worse) results



XRootD/Xcache in the context of Analysis Facilities

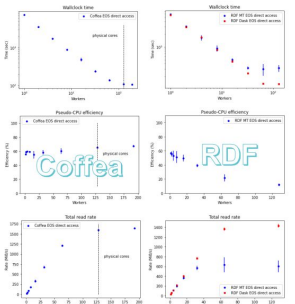
Local access

- **Scalability is excellent**
 - Some bottleneck appears for high numbers of workers
 - Overcommitting does not help for Coffea, but it increases throughput for RDF
 - Indication that Coffea is more CPU-constrained
- **The CPU efficiency comparably high**
 - I/O not a strong bottleneck
- **Local, fast SSD storage is always going to work well**
 - Aggregate read rates up to 3 GB/s



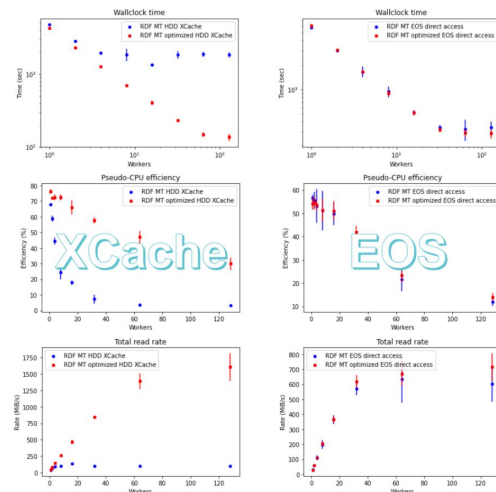
Direct access to EOS

- **Scalability still good when parallelism is via multiprocess**
 - RDF implicit multithreading does not perform well with xrootd and many threads
 - RDF via Dask is multiprocess, scales better
- **CPU efficiency practically constant around 60% with multiprocess parallelism**
 - I/O time is not negligible anymore but no bottlenecks
- **Two EOS instances tested**
 - EOSCMS by default, but CERNBOX produces similar (slightly worse) results

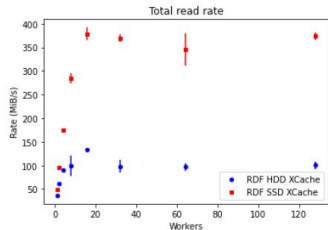
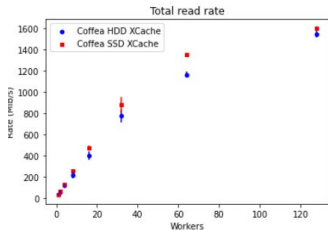
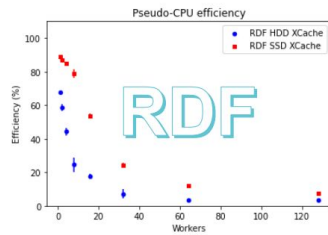
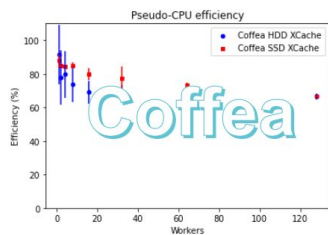
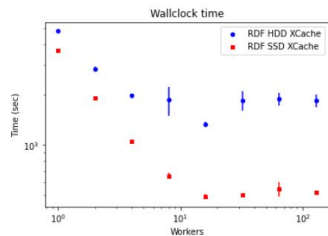
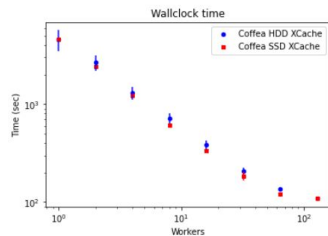


RDF + Xrootd performance optimization

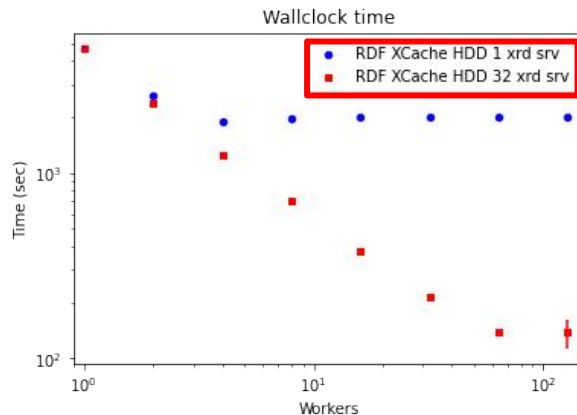
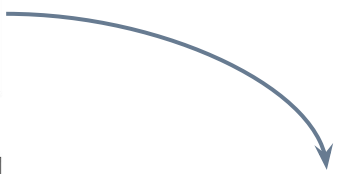
- **Scalability with ROOT multithreading and XCache can be improved**
 - XRD_PARALLELEVTLOOP=10 on the client largely improves Xrootd performance
 - Prevent the connection multiplexing by adding different client names to the file names
`root://client1@eoscms.cern.ch/eos/myfile.root`
- **Enormous impact when reading from XCache**
 - Obvious as it is a single server and multiplexing a big bottleneck
- **Effect negligible when reading from EOS**
 - Files already spread over hundreds of disk servers, multiplexing irrelevant



XRootD/Xcache in the context of Analysis Facilities



XCache server with internal clustering performs much better for ROOT multi-threaded analysis.



No “fake clients” trick needed here.

XRootD 5.6.0 Highlights

► XRootD 5.6.0 (cont.)

- Build System / C++
 - Modernization of CMake build system (requires CMake 3.16+)
 - Support building against musl libc (e.g. on Alpine Linux, Void Linux)
 - Make codebase C++17 ready (migration to C++17 as baseline in XRootD 5.7.0)
 - XrdCeph git submodule merged back into the main repository
 - Build option **ENABLE_CRYPTO** removed. OpenSSL always required with XRootD 5.x
- Testing and Continuous Integration
 - Updated continuous integration system based on GitHub Actions
 - New **xrd-docker** to automate building/running container-based tests
 - New **test.cmake** script added to automate configure/build/test cycle
- RPM/DEB Packaging
 - Rewrite of both DEB/RPM packaging to simplify building official repository
- Python bindings
 - Better support for Python 3.x, rewrite of build system in accordance with **PEP517**

XRootD 5.6.0 Highlights

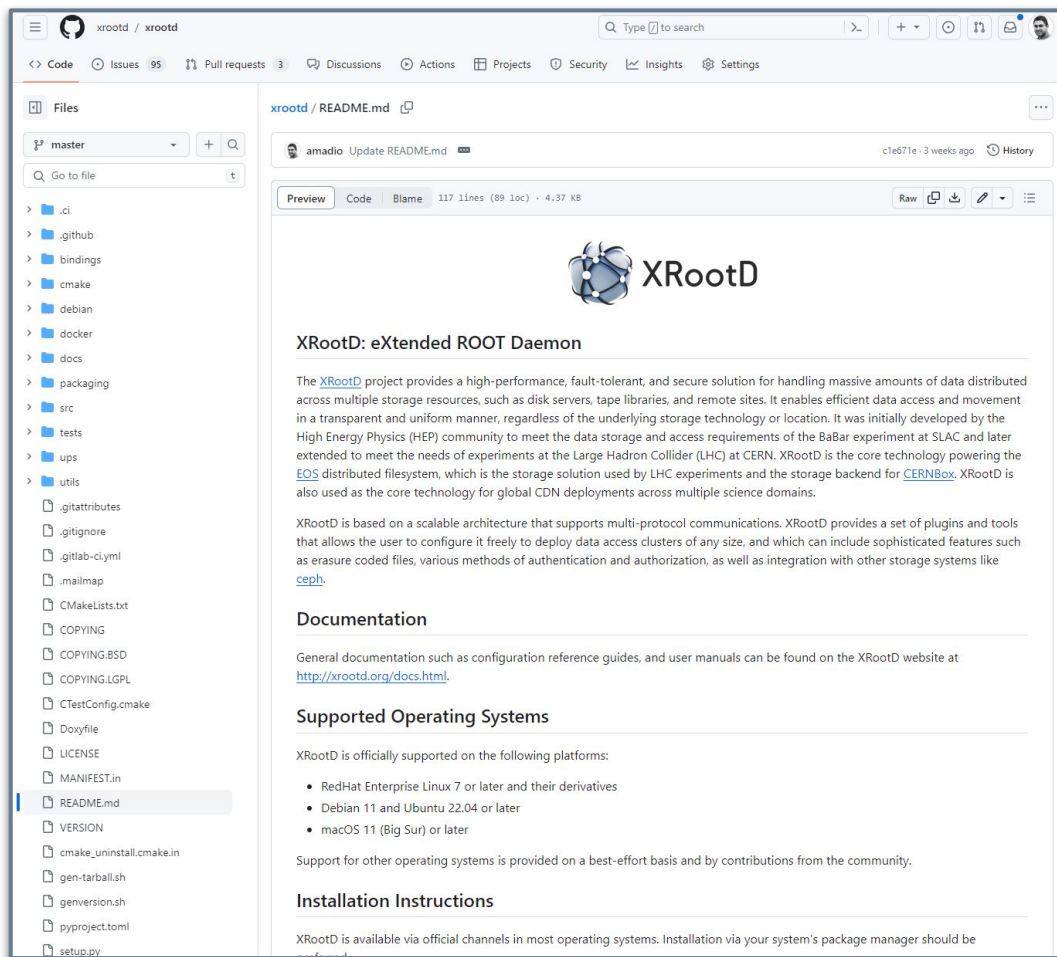
▶ XRootD 5.6.0 (cont.)

- XrdHttp
 - Include **Accept-Ranges** in **HEAD** response
 - Allow XRootD to return trailers indicating failure
 - Report cache object age for caching proxy mode
 - Return 405 instead of 500 error code on deletion of non-empty directory
 - Return 404 instead of 500 error code on **GET** request on non-existent file
- Plugins
 - GSI: Add option to display DN when it differs from entity name
 - ZTN: Allow to point to a token file using cgi **?xrd.ztn=tokenfile**
 - ZTN: Allow option **-tokenlib none** to disable token validation (used by EOS)
- Miscellaneous
 - Updated docs: **README.md**, **INSTALL.md**, **TESTING.md**, **CONTRIBUTING.md**
 - Sandboxing / hardening settings added to systemd service units (commented out)
 - Make output of **xrdcrc32c** tool consistent with **xrdadler32**

XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
 - Continuous Integration
 - RPM / DEB Packages
 - Python wheels
 - QEMU cross-platform
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>



The screenshot shows the GitHub repository for XRootD. The left sidebar displays the file tree for the 'master' branch, with 'README.md' selected. The main content area shows the 'README.md' file, which includes the XRootD logo, the title 'XRootD: eXtended ROOT Daemon', and several sections of text: 'The XRootD project provides a high-performance, fault-tolerant, and secure solution...', 'XRootD is based on a scalable architecture...', 'Documentation', 'Supported Operating Systems', and 'Installation Instructions'.

GitHub repository page for `xrootd` showing the `Actions` tab. The page displays a workflow named `QEMU` with a `workflow_dispatch` event trigger. A modal dialog is open, allowing the user to run the workflow from the `master` branch on a `fedora` OS with `s390x` architecture.

Actions New workflow

All workflows

Workflows

- CI
- DEB
- Python
- QEMU**
- RPM

Management

- Caches
- Runners

QEMU
QEMU.yml

Filter workflow runs

1 workflow run

This workflow has a `workflow_dispatch` event trigger. Run workflow

Use workflow from

- Branch: `master`
- OS *
`fedora`
- Architecture *
`s390x`

Run workflow

QEMU #1: Manually run by amadio

Files

master

Go to file

- .ci
- .github/workflows
 - Clyml
 - DEB.yml
 - QEMU.yml
 - RPM.yml
 - python.yml
- bindings
- cmake
- debian
- docker
- docs
- packaging
- src
- tests
- ups
- utils

xrootd / .github / workflows / QEMU.yml

Code Blame 57 lines (49 loc) · 1.16 KB

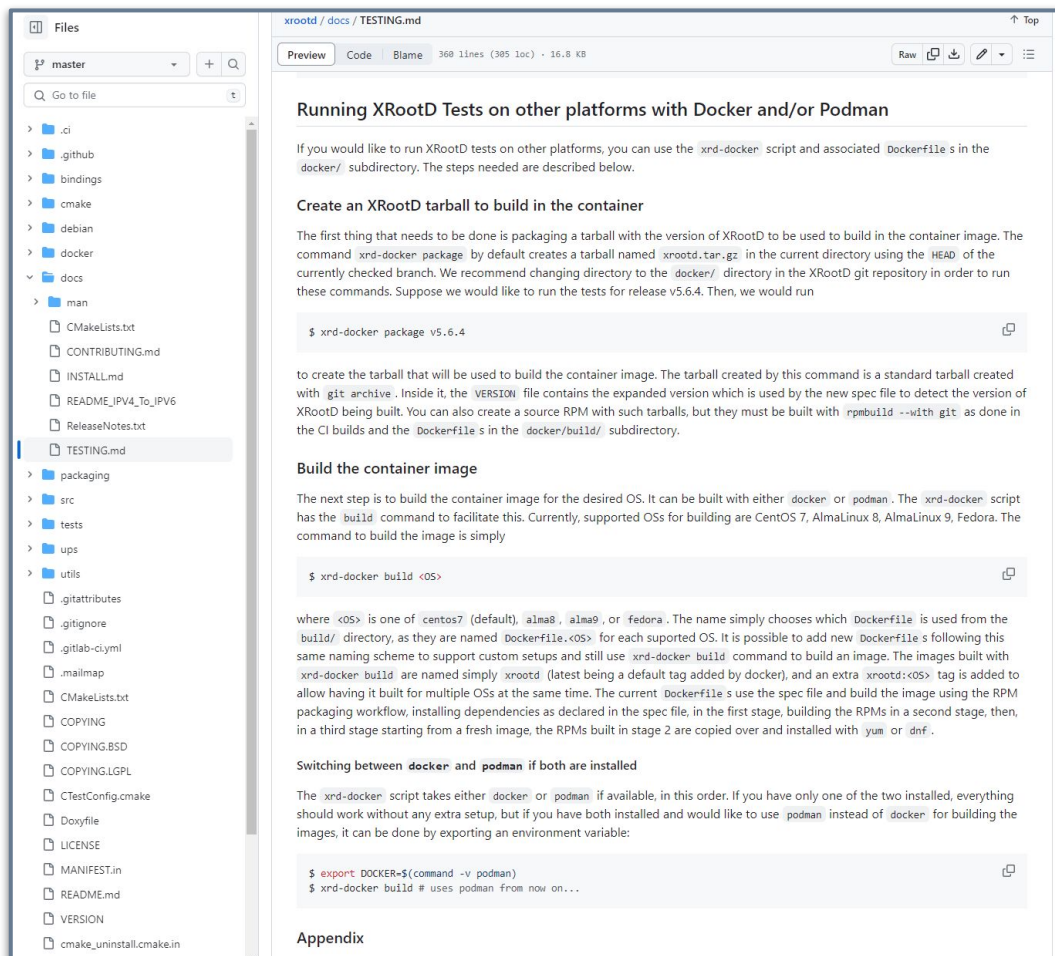
Raw Copy Download Edit

```
30
31 concurrency:
32   group: ${{ github.workflow }}-${{ github.ref }}-${{ inputs.os }}-${{ inputs.arch }}
33   cancel-in-progress: true
34
35 defaults:
36   run:
37     shell: bash
38
39 env:
40   DOCKER: podman
41
42 jobs:
43   buildx:
44     name: QEMU (${{ inputs.os }}-${{ inputs.arch }})
45     runs-on: ubuntu-latest
46
47     steps:
48     - name: Clone repository
49       uses: actions/checkout@v3
50       with:
51         fetch-depth: 0
52
53     - name: Setup QEMU for cross-building images
54       run: docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
55
56     - name: Cross-build container with docker/podman buildx
57       run: cd docker && ./xrd-docker buildx ${{ inputs.os }} ${{ inputs.arch }}
```

XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
 - Continuous Integration
 - RPM / DEB Packages
 - Python wheels
 - QEMU cross-platform
- ▶ CTest config/build/test script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>



Running XRootD Tests on other platforms with Docker and/or Podman

If you would like to run XRootD tests on other platforms, you can use the `xrd-docker` script and associated `Dockerfile`s in the `docker/` subdirectory. The steps needed are described below.

Create an XRootD tarball to build in the container

The first thing that needs to be done is packaging a tarball with the version of XRootD to be used to build in the container image. The command `xrd-docker package` by default creates a tarball named `xrootd.tar.gz` in the current directory using the `HEAD` of the currently checked branch. We recommend changing directory to the `docker/` directory in the XRootD git repository in order to run these commands. Suppose we would like to run the tests for release v5.6.4. Then, we would run

```
$ xrd-docker package v5.6.4
```

to create the tarball that will be used to build the container image. The tarball created by this command is a standard tarball created with `git archive`. Inside it, the `VERSION` file contains the expanded version which is used by the new spec file to detect the version of XRootD being built. You can also create a source RPM with such tarballs, but they must be built with `rpmbuild --with git` as done in the CI builds and the `Dockerfile`s in the `docker/build/` subdirectory.

Build the container image

The next step is to build the container image for the desired OS. It can be built with either `docker` or `podman`. The `xrd-docker` script has the `build` command to facilitate this. Currently, supported OSs for building are CentOS 7, AlmaLinux 8, AlmaLinux 9, Fedora. The command to build the image is simply

```
$ xrd-docker build <OS>
```

where `<OS>` is one of `centos7` (default), `alma8`, `alma9`, or `fedora`. The name simply chooses which `Dockerfile` is used from the `build/` directory, as they are named `Dockerfile.<OS>` for each supported OS. It is possible to add new `Dockerfile`s following this same naming scheme to support custom setups and still use `xrd-docker build` command to build an image. The images built with `xrd-docker build` are named simply `xrootd` (latest being a default tag added by `docker`), and an extra `xrootd:<OS>` tag is added to allow having it built for multiple OSs at the same time. The current `Dockerfile`s use the spec file and build the image using the RPM packaging workflow, installing dependencies as declared in the spec file, in the first stage, building the RPMs in a second stage, then, in a third stage starting from a fresh image, the RPMs built in stage 2 are copied over and installed with `yum` or `dnf`.

Switching between `docker` and `podman` if both are installed

The `xrd-docker` script takes either `docker` or `podman` if available, in this order. If you have only one of the two installed, everything should work without any extra setup, but if you have both installed and would like to use `podman` instead of `docker` for building the images, it can be done by exporting an environment variable:

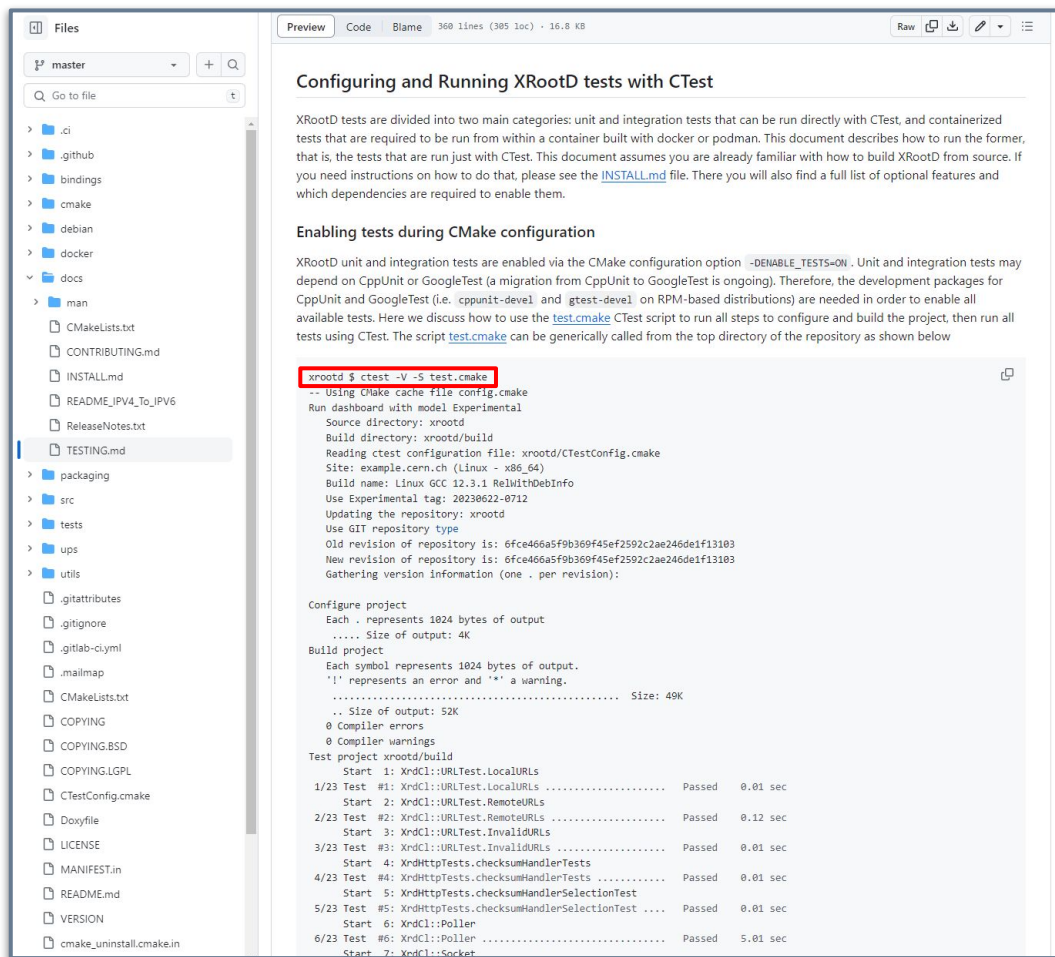
```
$ export DOCKER=$(command -v podman)
$ xrd-docker build # uses podman from now on...
```

Appendix

XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
 - Continuous Integration
 - RPM / DEB Packages
 - Python wheels
 - QEMU cross-platform
- ▶ CTest config/build/test script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>



The screenshot displays a GitHub repository interface. On the left, the file explorer shows the repository structure, with the `TESTING.md` file selected. On the right, a terminal window shows the execution of the `ctest` command. The terminal output includes the following information:

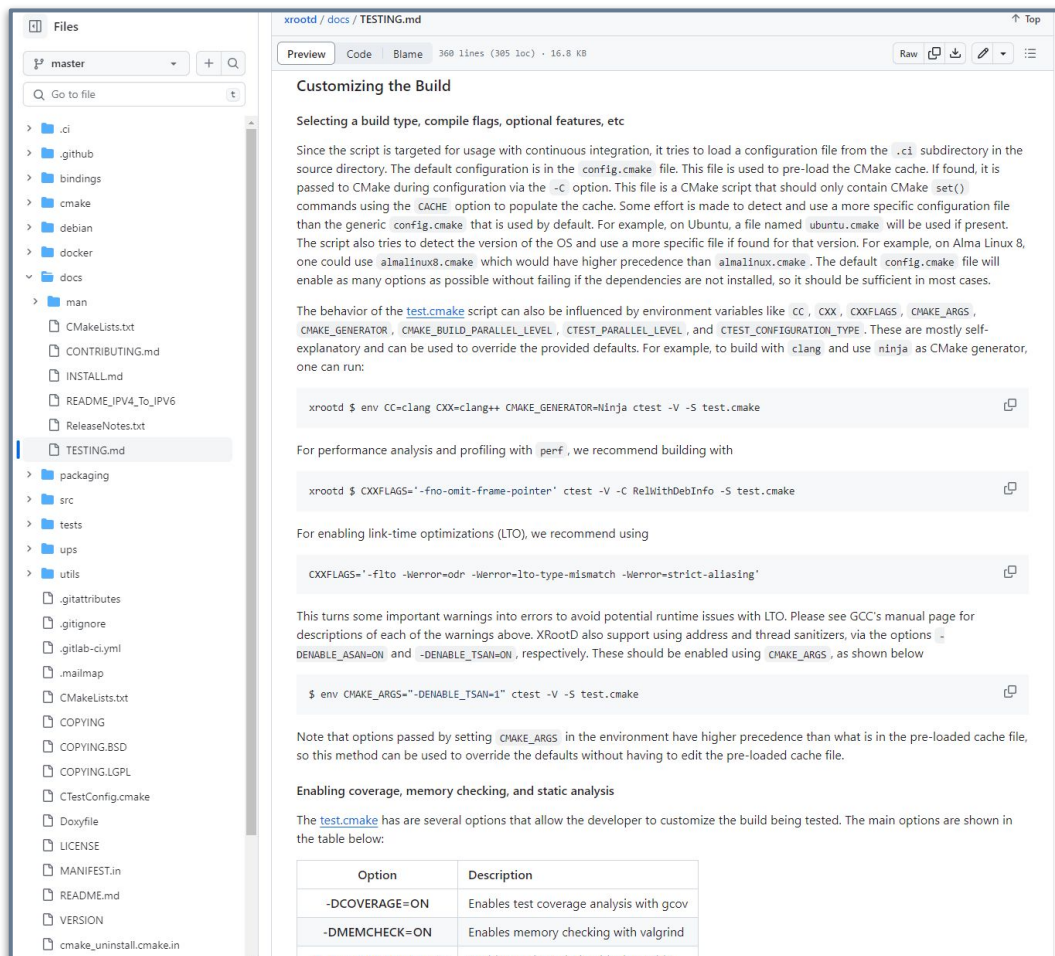
```
xrootd $ ctest -V -S test.cmake
-- Using CMake cache file config.cmake
Run dashboard with model Experimental
Source directory: xrootd
Build directory: xrootd/build
Reading ctest configuration file: xrootd/CTestConfig.cmake
Site: example.cern.ch (Linux - x86_64)
Build name: Linux GCC 12.3.1 RelWithDebInfo
Use Experimental tag: 20230622-0712
Updating the repository: xrootd
Use GIT repository type
Old revision of repository is: 6fce466a5f9b369f45ef2592c2ae246de1f13103
New revision of repository is: 6fce466a5f9b369f45ef2592c2ae246de1f13103
Gathering version information (one . per revision):

Configure project
Each . represents 1024 bytes of output
..... Size of output: 4K
Build project
Each symbol represents 1024 bytes of output.
'!' represents an error and '*' a warning.
..... Size: 49K
.. Size of output: 52K
0 Compiler errors
0 Compiler warnings
Test project xrootd/build
Start 1: XrdCl::URLTest.LocalURLs
1/23 Test #1: XrdCl::URLTest.LocalURLs ..... Passed 0.01 sec
Start 2: XrdCl::URLTest.RemoteURLs
2/23 Test #2: XrdCl::URLTest.RemoteURLs ..... Passed 0.12 sec
Start 3: XrdCl::URLTest.InvalidURLs
3/23 Test #3: XrdCl::URLTest.InvalidURLs ..... Passed 0.01 sec
Start 4: XrdHttpTests.checksumHandlerTests
4/23 Test #4: XrdHttpTests.checksumHandlerTests ..... Passed 0.01 sec
Start 5: XrdHttpTests.checksumHandlerSelectionTest
5/23 Test #5: XrdHttpTests.checksumHandlerSelectionTest .... Passed 0.01 sec
Start 6: XrdCl::Poller
6/23 Test #6: XrdCl::Poller ..... Passed 5.01 sec
Start 7: XrdCl::Socket
```


XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
 - Continuous Integration
 - RPM / DEB Packages
 - Python wheels
 - QEMU cross-platform
- ▶ CTest config/build/test script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>



The screenshot shows the GitHub interface for the file `docs/TESTING.md`. The left sidebar shows the repository structure with `docs/TESTING.md` selected. The main content area displays the README text, which is divided into sections: "Customizing the Build", "Selecting a build type, compile flags, optional features, etc", "Enabling coverage, memory checking, and static analysis", and a table of options.

Customizing the Build

Selecting a build type, compile flags, optional features, etc

Since the script is targeted for usage with continuous integration, it tries to load a configuration file from the `.ci` subdirectory in the source directory. The default configuration is in the `config.cmake` file. This file is used to pre-load the CMake cache. If found, it is passed to CMake during configuration via the `-C` option. This file is a CMake script that should only contain CMake `set()` commands using the `CACHE` option to populate the cache. Some effort is made to detect and use a more specific configuration file than the generic `config.cmake` that is used by default. For example, on Ubuntu, a file named `ubuntu.cmake` will be used if present. The script also tries to detect the version of the OS and use a more specific file if found for that version. For example, on Alma Linux 8, one could use `alma1linux8.cmake` which would have higher precedence than `alma1linux.cmake`. The default `config.cmake` file will enable as many options as possible without failing if the dependencies are not installed, so it should be sufficient in most cases.

The behavior of the `test.cmake` script can also be influenced by environment variables like `CC`, `CXX`, `CXXFLAGS`, `CMAKE_ARGS`, `CMAKE_GENERATOR`, `CMAKE_BUILD_PARALLEL_LEVEL`, `CTEST_PARALLEL_LEVEL`, and `CTEST_CONFIGURATION_TYPE`. These are mostly self-explanatory and can be used to override the provided defaults. For example, to build with `clang` and use `ninja` as CMake generator, one can run:

```
xrootd $ env CC=clang CXX=clang++ CMAKE_GENERATOR=Ninja ctest -V -S test.cmake
```

For performance analysis and profiling with `perf`, we recommend building with

```
xrootd $ CXXFLAGS="-fno-omit-frame-pointer" ctest -V -C RelWithDebInfo -S test.cmake
```

For enabling link-time optimizations (LTO), we recommend using

```
CXXFLAGS="-flto -Werror-odr -Werror-lto-type-mismatch -Werror-strict-aliasing"
```

This turns some important warnings into errors to avoid potential runtime issues with LTO. Please see GCC's manual page for descriptions of each of the warnings above. XRootD also support using address and thread sanitizers, via the options `-DENABLE_ASAN=ON` and `-DENABLE_TSAN=ON`, respectively. These should be enabled using `CMAKE_ARGS`, as shown below

```
$ env CMAKE_ARGS="-DENABLE_TSAN=1" ctest -V -S test.cmake
```

Note that options passed by setting `CMAKE_ARGS` in the environment have higher precedence than what is in the pre-loaded cache file, so this method can be used to override the defaults without having to edit the pre-loaded cache file.

Enabling coverage, memory checking, and static analysis

The `test.cmake` has several options that allow the developer to customize the build being tested. The main options are shown in the table below:

Option	Description
<code>-DCOVERAGE=ON</code>	Enables test coverage analysis with <code>gcov</code>
<code>-DMEMCHECK=ON</code>	Enables memory checking with <code>valgrind</code>

XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
 - Continuous Integration
 - RPM / DEB Packages
 - Python wheels
 - QEMU cross-platform
- ▶ CTest config/build/test script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>

Continuous		[view timeline]										
		Update	Configure		Build		Test					
Site	Build Name	Revision	Error	Warn	Error	Warn	Not Run	Fail	Pass	Time	Start Time	
GitHub Actions (xrootd)	Fedora Linux 39 GCC 13.2.1 RelWithDebInfo Ninja (devel)	575319	0	0	0	0	0	0	107	2m 18s	22 hours ago	
GitHub Actions (xrootd)	CentOS Linux 7 GCC 7.3.1 RelWithDebInfo (devel)		0	0	0	0	0	0	105	2m 23s	22 hours ago	
GitHub Actions (xrootd)	AlmaLinux 8.9 GCC 8.5.0 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 21s	22 hours ago	
GitHub Actions (xrootd)	macOS 12.7.3 AppleClang 14.0.0.14000029 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 47s	22 hours ago	
GitHub Actions (xrootd)	AlmaLinux 9.3 GCC 11.4.1 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 21s	22 hours ago	
GitHub Actions (xrootd)	Ubuntu 22.04.4 LTS GCC 11.4.0 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 19s	22 hours ago	
GitHub Actions (xrootd)	Alpine Linux v3.19 GCC 13.2.1 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	107	2m 16s	22 hours ago	
GitHub Actions (xrootd)	Ubuntu 22.04.4 LTS Clang 14.0.0 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 19s	22 hours ago	
GitHub Actions (xrootd)	Fedora Linux 39 GCC 13.2.1 RelWithDebInfo Ninja (devel)	a8fd3	0	0	0	0	0	0	107	2m 18s	Mar 13, 2024 - 01:54 UTC	
GitHub Actions (xrootd)	CentOS Linux 7 GCC 7.3.1 RelWithDebInfo (devel)		0	0	0	0	0	0	105	2m 23s	Mar 13, 2024 - 01:54 UTC	
GitHub Actions (xrootd)	AlmaLinux 8.9 GCC 8.5.0 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	2m 22s	Mar 13, 2024 - 01:53 UTC	
GitHub Actions (xrootd)	AlmaLinux 9.3 GCC 11.4.1 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	2m 21s	Mar 13, 2024 - 01:53 UTC	
GitHub Actions (xrootd)	Ubuntu 22.04.4 LTS Clang 14.0.0 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	2m 19s	Mar 13, 2024 - 01:53 UTC	
GitHub Actions (xrootd)	Ubuntu 22.04.4 LTS GCC 11.4.0 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	2m 19s	Mar 13, 2024 - 01:53 UTC	
GitHub Actions (xrootd)	macOS 12.7.3 AppleClang 14.0.0.14000029 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	5m 44s	Mar 13, 2024 - 01:53 UTC	
GitHub Actions (xrootd)	Alpine Linux v3.19 GCC 13.2.1 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	107	2m 17s	Mar 13, 2024 - 01:53 UTC	

Items per page: All

CDash v3.3.0-rc2-6-ga975ec23d © Kitware | Report problems | View as JSON | 0.06s (0.05s)
Current Testing Day 2024-03-14 | Started at 01:00 UTC

Next Release: XRootD 5.7.0

- ▶ Move to C++17 standard as baseline
- ▶ Remove **using namespace std;** from all headers and source files
- ▶ Allow changing C++ standard via **CMAKE_CXX_STANDARD**
- ▶ Update min/default RSA key bits to 2048
- ▶ Full migration of testing infrastructure to GoogleTest (CppUnit tests removed)
- ▶ New **only-if-cached** directive for Xcache
- ▶ New **tpc.route** option to force connecting back via same IP on HTTP-TPC
- ▶ Disallow renegotiation for TLSv1.2 and earlier

Future Plans

- ▶ Continue to extend testing coverage
 - HTTP, TPC, XCache, authentication methods (host, krb5, gsi, tokens, sss, unix)
- ▶ Resolve at least ~20% of outstanding enhancements per year
- ▶ Support for alternative protocols (e.g. SFTP, an EGI request)
- ▶ Native Oauth2 plugin
 - Enable using CERN SSO tokens with XRootD/EOS
- ▶ SSH Authentication
 - SSH keys/certificates as possible alternative to X509 certificates
- ▶ Better, multi-protocol integration with FTS
 - Reduced interface needed for file transfers
- ▶ Range cloning of files to allow updating of EC files
- ▶ Plugin for handling RUCIO datasets within ZIP archives

XRootD Packaging and Distribution

- ▶ New official RPM repositories, include Alma 9, Fedora
<https://xrootd.slac.stanford.edu/dload.html#official-rpm-repositories>
- ▶ RPM and DEB packaging scripts available in official GitHub repository
- ▶ XRootD is available via official channels in most distributions already
 - Alma, Arch, CentOS, Debian, Fedora, Gentoo, Manjaro, Raspbian, Rocky, Ubuntu, etc
- ▶ XRootD is also available on repositories that work across other OSs
 - Homebrew, macPorts, Nix, Spack, conda, etc
- ▶ Installation via official channels **strongly encouraged**



XRootD

XRootD 5.5.x Highlights

▶ XRootD 5.5.2

- Enable ZTN authentication with macaroons-based tokens
- Extend number of parallel copies from 4 to 128 (--parallel option to xrscp)

▶ XRootD 5.5.3

- Support user-provided script for computing checksums

▶ XRootD 5.5.4

- ZTN plugin enabled by default
- Fixes for authentication failures across daylight savings change
- Support certificates with dates in year 2049 or later

▶ XRootD 5.5.5

- Enable XrdClProxy plugin to work with pgRead
- Fix creation of zip archives with many entries
- Fix for mixing of reused file handles coming from external table (seen on EOS AMS)

XRootD 5.6.x Highlights

▶ XRootD 5.6.1

- Use kernel provided uuid on macOS
- Set **RPATH** that works for binaries and libraries on macOS

▶ XRootD 5.6.2

- HTTP: Fix chunked **PUT** creating empty files
- SciTokens: Update maximum header size and line length in INI files
- Fix template for default ZTN token location
- Change the thread-id returned to OpenSSL 1.0.x to improve performance
- Insert CRLs containing critical extensions at the end of the bundle
- XrdClHttp: Add pgWrite support to the HTTP client plugin
- Export readv comma separated limits via **XRD_READV_LIMITS** environment variable
- Python: Allow build customization via environment variables (e.g. **CXX**, **CXXFLAGS**)
- Fix promotion of **root://** URLs to use TLS encryption (bug introduced in 5.6.0)

XRootD 5.6.x Highlights

▶ XRootD 5.6.3

- Export project version in **XRootDConfig.cmake** module
- Create environment file within **xrd.adminpath**
- Return an error if **xrdfs** rm fails to delete any file
- Initial packet marking support in HTTP TPC

▶ XRootD 5.6.4

- Use full certificate chain for verification
- Migrate tests to GoogleTest and run without containers
- Add integrity check for headers and fix header dependency issues
- Fixes on SPARC architecture and GNU/Hurd (external contributions)
- Fix crash on **pss.origin** directive without specifying a port (uses protocol default)

▶ XRootD 5.6.5

- Support GCC 14
- Export project version in **XRootDConfig.cmake** module

XRootD 5.6.x Highlights

▶ XRootD 5.6.6

- Use full certificate chain for verification
- Migrate tests to GoogleTest and run without containers
- Add integrity check for headers and fix header dependency issues
- Fixes on SPARC architecture and GNU/Hurd (external contributions)
- Fix crash on **pss.origin** directive without specifying a port (uses protocol default)

▶ XRootD 5.6.7

- Fix crash at teardown when using copies with multiple streams
- Fix TPC initialization to take into account control stream (was always using 2 streams before)

▶ XRootD 5.6.8

- Only claim to be TLS capable if TLS initialization succeeds (**--notlsok** no longer needed)
- Create **CDash** dashboard for XRootD and enable submissions in **test.cmake**
- Fix build on FreeBSD

XRootD 5.6.x Highlights

▶ XRootD 5.6.9

- Python
 - Fix iteration over a file with Python3
 - Fix crash with raw strings in prepare call
- HTTP TPC
 - Fix 500 server response code if X-Number-Of-Streams > 100
- XrdSciTokens
 - Add stat permissions to create, modify and write operations
 - Allow creation of parent directories if necessary
 - Fix bug when scope includes basepath or /



XRootD