Status and features of the XRootD 4.0.0 release

Łukasz Janyst
on behalf of the XRootD collaboration

CERN IT Department
CH-1211 Genève 23
Switzerland
www.cern.ch/it

XRootD

ALICE T1/T2 Workshop
Lyon, 05.05.2013

Outline

- Important new features
- Impact on the existing code
- Status

New client

- Complete re-write with emphasis on scalability and performance
- Decouple the caching
- Overall maintainability
- Will be platform for future development
- Current client only patched for major issues

- Available in 3.3.0 (without ABI compat guarantees)
- To be stabilized in 4.0.0

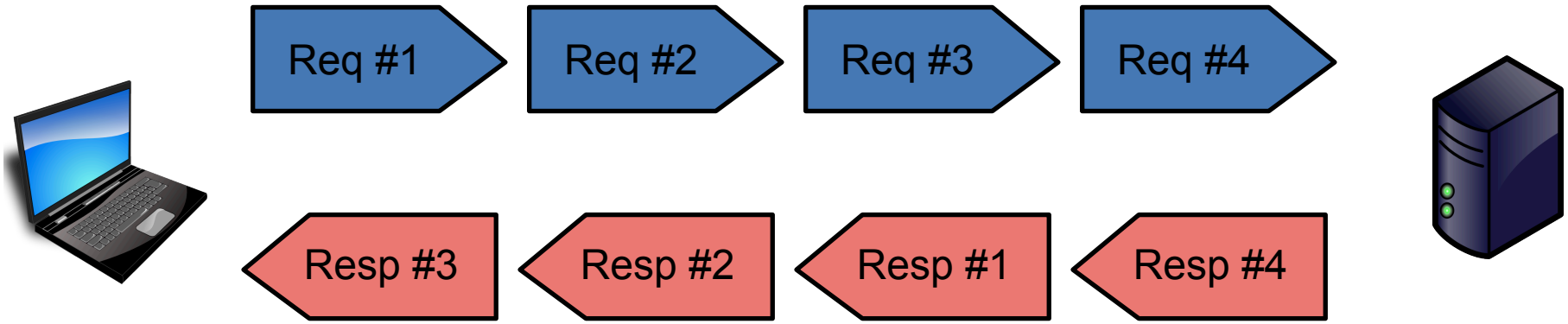
Asynchronous

- All of the xroot protocol requests implemented as asynchronous methods
- The calls queue the request and return, **never block**

```
XRootDStatus File::Open( const std::string &url,
                         OpenFlags::Flags   flags,
                         Access::Mode       mode,
                         ResponseHandler   *handler,
                         uint16_t           timeout )
```

- The response handler is called when the response is ready
- No need to have cache to handle buffers
- Synchronous calls implemented using async ones

Async vs. stream multiplexing



- The XRootD protocol supports virtual streams
- There may be many requests outstanding and the server may respond in the order it chooses
- The new client handles responses as soon as they come calling the user call-back function

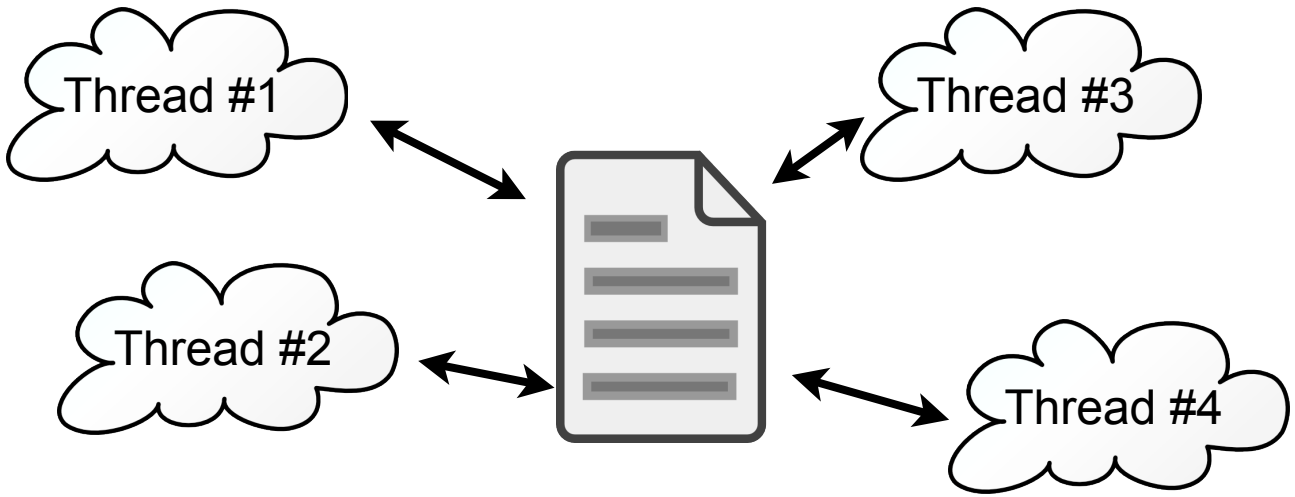
Performance - metadata ops


```
]==> time xrd metaman dirlist /data/bigdir > /dev/null
1.58s user 1.94s system 4% cpu 1:18.09 total

]==> time xrdfs metaman ls -l /data/bigdir > /dev/null
1.26s user 0.46s system 64% cpu 2.678 total
```

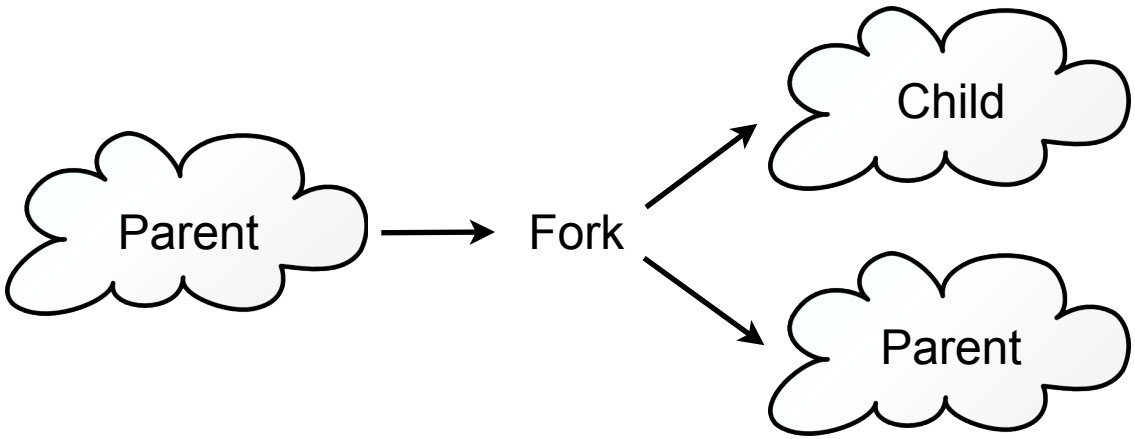
- List a directory of 40k files spread across 4 servers
- Link: 100 Mbps, round-trip 1.8 ms

Thread safe/aware



- File and FileSystem objects can be safely accessed from multiple execution threads
- Internally uses a worker thread pool to handle call-backs

Fork safe



- Can handle forking even when the IO operations are in progress
- File and FileSystem objects remain valid in both parent and child
- The operations in the parent continue after the fork
- The objects in the child will run recovery procedure (like in the case of a broken connection)

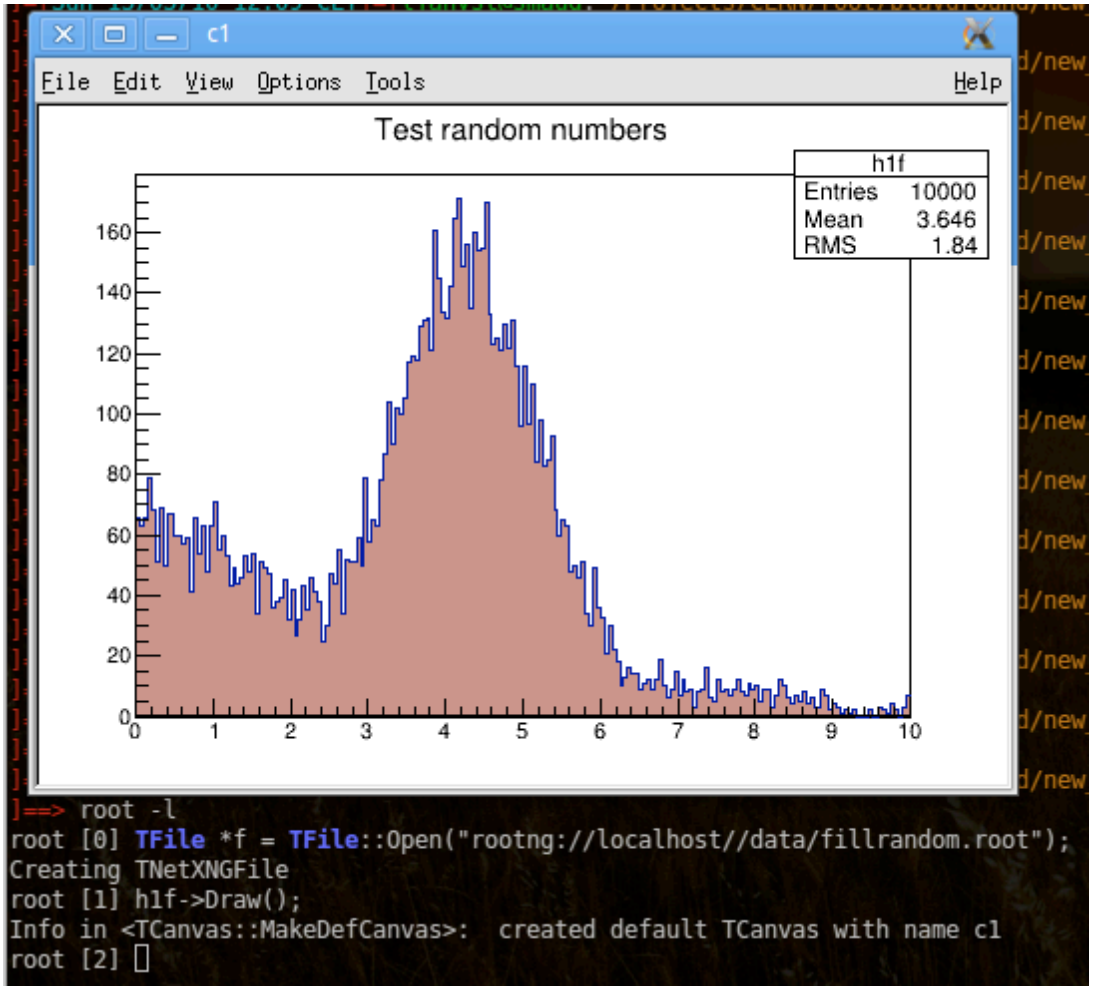
User classes

- **XrdCl::FileSystem** for meta-data requests
  - mkdir, rmdir, query, locate, move truncate, chmod, ping, stat...
- **XrdCl::File** for data operations
  - read, write, readv...
- Redesigned API, not backwards compatible but rarely used directly (interfaced by ROOT)

Command line

- **xrdcopy** (replacement for **xrdcp**) - backwards compatible interface, drop-in replacement, heavily used
- **xrdfs** (replacement for **xrd**) - cleanups to the interface, rarely used

ROOT plug-in



- The plugin code is finished and is being tested
- It's fairly simple, mostly 1-to-1 mapping of methods
- Due to some issues with ROOT garbage collector needs the 3.3.3 release

ROOT plugin

- Both new and old plug-in can co-exist at runtime
- For **testing** purposes selection can be done:
  - by changing the file URL (ie. **root://** to **rootng://**) or
  - setting the **XRD_CLIENT** environment variable or
  - setting a variable in a **.rootrc** file or
  - using **gEnv**
- In the final version the new plugin should replace the old one as the default

Proxy readv support

- Current proxy unrolls readv's and forwards them as ordinary reads
  - this may severely impact performance
- Solved by integrating readv interfaces
  - allows proxies to pass-through readv's
  - Impact: plugins need to be recompiled
- Code contributed by **Brian Bockelman** (CMS)

IPv6

- We have decided to do a **clean transition**
  - more maintainable than a “patch”
  - based on new **IP agnostic framework**
- Impact:
  - Network sensitive server plugins need to change
    - changes are apparent and minimal
  - No changes client-side

- One server daemon for both stacks
- The client will choose the protocol **preferring IPv6** if available

Internal Bridge Protocol

- Makes it possible to **map protocols**
  - protocol x to xroot and execute
  - xroot responds back using x protocol
- Allows for plugging-in other protocols while still leveraging xroot features
  - monitoring, sessions, security, etc.
- **Fabrizio Furano** works on a bridged **HTTP** plugin

HTTP to replace XRootD?

- It's just an **addition** that opens new service possibilities:
  - Full file downloads
  - Possible SRM replacement interface
- Will HTTP replace the XRootD protocol?
  - Current HTTP definitely not!
- Why? Mainly because:
  - does not allow for out-of-order or interleaved responses
    - reasonable performance only for large data buffers
  - has large headers
    - protocol overhead is significant
  - sessions and authentication
  - parsing and interpreting lots text is CPU intensive

Can HTTP be fixed?

- Google is proposing **SPDY**
  - https://en.wikipedia.org/wiki/SPDY
- Microsoft is proposing **Speed+Mobility**
  - https://en.wikipedia.org/wiki/Microsoft_SM
- Each addresses bandwidth and latency issue
- Neither really addresses CPU issues
  - Servers are cheap in the cloud
  - SPDY uses header compression - needs more CPU

- IETF is working on **HTTP 2.0** with SPDY being the initial draft

But HTTP is everywhere!

- Naively, yes. But:
  - robust usage requires custom clients negating the “everywhere” concept, just ask Amazon or Google
  - the Internet Explorer effect - one badly behaving wide-spread client can adversely affect the server performance and maintainability

Configurable IO Throttling

- Implemented as a plug-in
  - minimal impact on existing code paths
- Allows you to control the file-based transfer rate by connection
- Useful for limiting bandwidth of “external” (ie. not local) connections when federating storage sites.

- Contributed by **Brian Bockelman** (CMS)

Development

- The official code repository has moved to GitHub:
  - https://github.com/xrootd/xrootd
  - we also use GitHub for bug tracking and contribution review
- We have implemented library compatibility checks in the incremental build system
- We are paying attention to C++11 compatibility

Conclusion

- The XRootD base code has seen plenty of performance improvements and new features
- The experiment client-side code is mostly shielded by ROOT - typically no changes needed
- The server side-code needs minimal changes and recompilation in order to work
- We expect to release the code in the middle of July

Questions

Thanks for your attention!

Questions? Comments?