ROOT Developer Retreat: ROOT I/O

18 December 2019
ROOT I/O

- > 1 EB of physics data in ROOT format
- I/O is one of ROOT’s killer features: fastest analysis I/O in class
- Feature-rich: seamless C++ serialization, comprehensive set of compression algorithms, efficient remote I/O, schema evolution, etc.
- But also:
  - a. not directly compatible with industry standard big data tooling which results in conversion stacks popping up (ROOT to numpy, Spark, S3, etc.)
  - b. Programming low-level I/O is (inherently?) difficult; most users would probably prefer to not directly use I/O classes
- RNTuple status and plans
  - a. Solid class architecture, powerful enough for simple ntuple tasks
  - b. Significant performance improvement seen across the board, particularly on ultra-fast storage
  - c. Now needs to be consolidated to accommodate first users (say between 1 and 3 adventurous, relevant ones)
ROOT I/O Person Power in 2020

- Philippe [50 %]
- Oksana [50 %]
- Jakob [50 %]
- A new fellow as of summer 2020
- Yungsong Wang (LBL) for TMPIFileMerger
- Vincenzo as of February [possibly a small fraction of his time]

- Bernhard (Helmholtz Dresden) on research of GPU friendly event data models
- A few student months (summer student, GSoC, etc.)
- Plus contributions from Brian, Jim, Danilo, CERN storage team, and others

There is interest from beyond the ROOT core team in I/O. We should try to leverage on that by suggesting proper “satellite tasks”.
ROOT I/O Areas of Work

1. **TTree**: maintenance, user support, bug fixes, support for critical new features

2. **RNTuple**: clamp the binary format, acquire first users

3. **Cross-cutting issues**: compression, error handling, benchmarks

---

**Task classification**

*italics == unassigned task*

- **Difficulty**: starter project → advanced → R&D
- **Urgency**: nice to have → important → essential (target <= v6.24)
- **Progress**: planned → well underway → almost done
TTree: Core Business

1. **User support and bug fixes (forum, bug tracker, etc.)**
   Difficulty: advanced  
   Urgency: essential

2. **Bug fixes**
   Difficulty: advanced  
   Urgency: essential

3. **Thread-safety and performance improvements**: including TBufferFile
   Difficulty: advanced  
   Urgency: essential  
   Progress: planned  
   ~2 FTME

4. **Advance C++ type support**: shared_ptr, optional, variant, array
   (partially uncovered)
   Difficulty: advanced  
   Urgency: important  
   Progress: planned  
   ~4.5 FTME
TTree: Integration

1. **TMPIFile and Merger**
   - Difficulty: starter project
   - Urgency: important
   - Progress: almost done
   - ~1 FTME

2. **RDataFrame Bulk I/O DataSource** *(planned to be addressed more fundamentally in RDF)*
   - Difficulty: advanced
   - Urgency: important
   - Progress: planned
   - ~2 FTME

3. **Direct path TTree → Bulk I/O → numpy arrays**: provides a “fast” connection between ROOT I/O and the Python world
   - Difficulty: advanced
   - Urgency: nice to have
   - Progress: planned
   - ~1 FTME
**TTree: Stretch Goals**

1. **Improve performance of TBranchFile**: remove virtual function calls  
   Difficulty: advanced  
   Urgency: important  
   Progress: planned  
   ~1 FTME

2. **I/O of interpreted classes**: avoid having to spell out all used class template instances  
   (somewhat blocker for ROOT7 histograms)  
   Difficulty: advanced  
   Urgency: important  
   Progress: planned  
   ~1 FTME

3. **I/O of interpreted collections**: allow streaming of all interpreted classes  
   For experiment relying heavily on class template, including potentially ROOT v7  
   Difficulty: advanced  
   Urgency: important  
   Progress: planned  
   ~2 FTME

4. **Double32_t improvements**, customization of vector<Double32_t>, similar feature for integer  
   Difficulty: advanced  
   Urgency: nice to have  
   Progress: planned  
   ~1+2 FTME
RNTuple Core Business

1. **Finalise the binary file format**: including a specification
   Difficulty: advanced  
   Urgency: essential  
   Progress: well underway  
   ~2 FTME

2. **Finalise low-level ROOT container reader and writer**, reading without libCore dependency
   Difficulty: advanced  
   Urgency: essential  
   Progress: well underway  
   ~1 FTME

3. **Asynchronous cluster cache**: RNTuple’s I/O scheduler
   Difficulty: advanced  
   Urgency: essential  
   Progress: almost done  
   ~1 FTME

4. **I/O type casting**: relax requirement to read with exact same type that was used for writing
   (e.g. read any collection into an std::vector, perhaps read float in double, etc.)
   Difficulty: advanced  
   Urgency: essential  
   Progress: planned  
   ~2.5 FTME
RNTuple Usability and Performance I/II

1. **Friends and chains**: virtual storage backends
   - Difficulty: starter project
   - Urgency: essential
   - Progress: almost done
   - ~0.5 FTME

2. **Interactive data inspection**: TBrowser integration, schema printing, printing of entries, etc.
   - Difficulty: starter project
   - Urgency: essential
   - Progress: almost done
   - ~0.5 FTME

3. **TTree to RNTuple converter**: disk-to-disk conversion as in hadd
   - Difficulty: starter project
   - Urgency: essential
   - Progress: planned
   - ~2 FTME

4. **RNTuple independent file format reader into numpy**: facilitates finding early adopters and validates the file format specification
   - Difficulty: advanced
   - Urgency: important
   - Progress: planned
   - ~4 FTME
RNTuple Usability and Performance II/II

5. **Efficient RDF integration:** aim for a penalty factor < 2 compared to hand-optimised code
   Difficulty: R&D          Urgency: important          Progress: planned          ~6 FTME

6. **User-facing bulk API:** retrieve spans of entries
   Difficulty: advanced      Urgency: important      Progress: planned          ~1 FTME

7. **Object store backend:** store pages and clusters directly in S3, DODAS, or Ceph
   Difficulty: starter project      Urgency: important      Progress: planned          ~4 FTME

8. **RNTuple as a RooFit backend**
   Difficulty: starter project      Urgency: nice to have      Progress: planned          ~2 FTME
RNTuple Overflow Items / External Contributions

1. **Fine-grained multi-threading**: concurrent reading on shared cluster buffers
   - Difficulty: R&D
   - Urgency: important
   - Progress: planned
   - ~4 FTME

2. **Fast merging**: concatenate RNTuple files without rewriting pages
   - Difficulty: starter project
   - Urgency: important
   - Progress: planned
   - ~3 FTME

3. **Direct data exchange with Apache Arrow**
   - Difficulty: advanced
   - Urgency: nice to have
   - Progress: planned
   - ~3 FTME

4. **Stand-alone RNTuple library**
   - Difficulty: starter project
   - Urgency: nice to have
   - Progress: planned
   - ~ 1.5 FTME
Error Handling

1. Add infrastructure for exceptions to ROOT
   Difficulty: starter project  Urgency: essential  Progress: almost done  ~0.5 FTME

2. Use exceptions in RRawFile and RNTuple
   Difficulty: advanced  Urgency: essential  Progress: planned  ~0.25 FTME

3. Validate crash recovery in RNTuple
   Difficulty: starter project  Urgency: important  Progress: planned  ~2 FTME
ROOT Compression Library/Engine

1. Finish compression settings interface overhaul
   Difficulty: advanced   Urgency: important   Progress: planned   ~1 FTME

2. Refactor RZip library to make easier compression settings and debugging
   Difficulty: advanced   Urgency: important   Progress: planned   ~1 FTME

3. Investigate possibility to add compression algorithms via plugins
   Difficulty: advanced   Urgency: nice to have   Progress: planned   ~1 FTME
ROOT Lossless Compression Algorithms

1. Investigate further ZSTD compression dictionaries and testing for RNTuple and TTree (potential to gain space and time in a large set of CMS cases)
   Difficulty: advanced  
   Urgency: nice to have  
   Progress: planned  
   ~3 FTME

2. Investigate ZSTD byte-stream compression and test it for RNTuple and TTree
   Difficulty: advanced  
   Urgency: important  
   Progress: planned  
   ~3 FTME

3. Add LZ4 + Bitshuffle filter to a list of ROOT supported compression algorithms (and to decide how it should be shipped as a part of LZ4 or as a separate compression algorithm)
   Difficulty: advanced  
   Urgency: important  
   Progress: planned  
   ~3 FTME

4. Investigate LZ4 + Bitshuffle for RNTuple
   Difficulty: advanced  
   Urgency: important  
   Progress: planned  
   ~3 FTME
ROOT Lossy Algorithms (Floating Point)

1. Investigate lossy compression algorithms (e.g. zfp, SZ)
   Difficulty: R&D                  Urgency: important      Progress: planned          ~3 FTME

2. Incorporate lossy compression engine (Accelogic)
   Difficulty: advanced             Urgency: important      Progress: planned          ~ 1 FTME
Investigation of ROOT I/O Performance

1. **Finish performance continuous testing for ROOT I/O critical parts**: especially for RNTuple
   Difficulty: advanced         Urgency: important         Progress: planned         ~2 FTME

2. **Multi-client / shared storage performance behavior** in collaboration with XRootD and EOS teams
   Difficulty: advanced         Urgency: important         Progress: planned         ~1 FTME

3. **Define and monitor a list of potential ROOT I/O bottlenecks/overheads**
   Difficulty: advanced         Urgency: nice to have         Progress: planned         ~3 FTME

4. **Collect central set of ROOT sample files** of various use cases and sizes (possibly with agreements that these samples will be available only for I/O research related purposes)
   Difficulty: advanced         Urgency: nice to have         Progress: planned         ~3 FTME
Final Remarks

- We have a window of opportunity for a big leap forward in the ROOT I/O: LHC experiment needs for run 3 and 4 align with additional R&D resources and with a solid roadmap.

- There is more work than hands on the keyboard: we need to be smart about task prioritization, accommodation of external contributions, and expectation management.

- We have to reconcile taking care of the production code base with new developments.