Integration of RNTuple in ATLAS Athena

Florine de Geus[1,2] Javier Lopez-Gomez[1] Jakob Blomer[1] Marcin Nowak[3] Peter van Gemmeren[4]

[1]CERN [2]University of Amsterdam [3]Brookhaven National Laboratory [4]Argonne National Laboratory

CHEP 2023 – Norfolk, U.S.

May 8, 2023

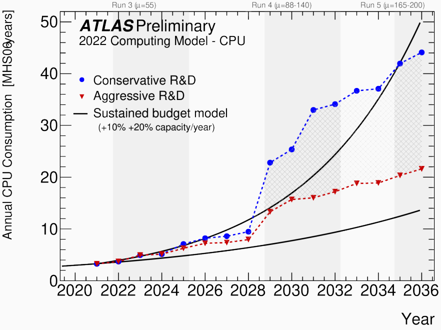
Background and motivation

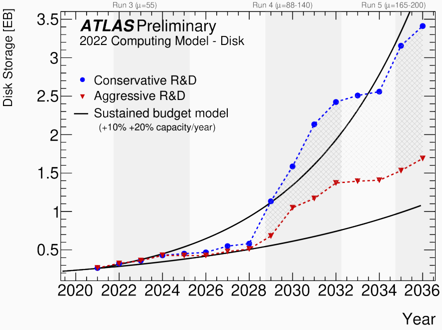
- **Athena:** ATLAS experiment software for data and MC processing
- For Run 3, **DAOD_PHYS** has been the common ATLAS-wide analysis format
  - Produced by deriving primary AODs resulting from data/MC reconstruction
- For Run 4, **DAOD_PHYSLITE** will additionally be used
  - Centrally calibrated, which means it needs to store fewer variables

ℹ **(D)AOD:** (Derived) Analysis Object Data

Background and motivation

HL-LHC: (even) more data to store and process





source

RNTuple: evolution of ROOT's TTree columnar data storage

Check out the *previous talk* *for more*

Getting RNTuple in shape for Athena

- Collection Proxies
  - Support for user-defined classes that behave as collections. These have an associated “collection proxy” that provides access to collection’s elements
- Read rules
  - Act on standard ROOT I/O customization rules (i.e., `#pragma read`)
  - Enables custom post-read callbacks
- Late model extension
  - Allows for on-demand extension of RNTuple model with new fields after some entries have been written using the initial schema
  - Required for adding dynamic attributes during the derivation step

With these additions, RNTuple-based DAOD_PHYS(LITE) production is fully supported in Athena

RNTuple for ATLAS DAOD_PHYS

Two central questions:

1. How does RNTuple perform compared to TTree?
2. What else is needed for RNTuple to be fully adopted by Athena?

RNTuple for ATLAS DAOD_PHYS

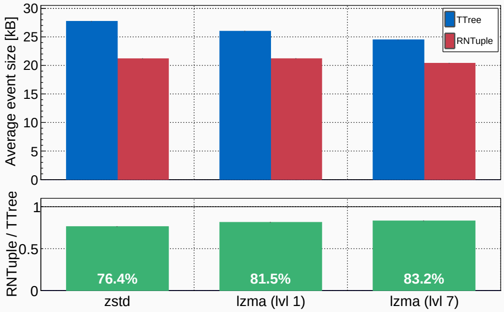
Two central questions:

1. How does RNTuple perform compared to TTree?
2. What else is needed for RNTuple to be fully adopted by Athena?

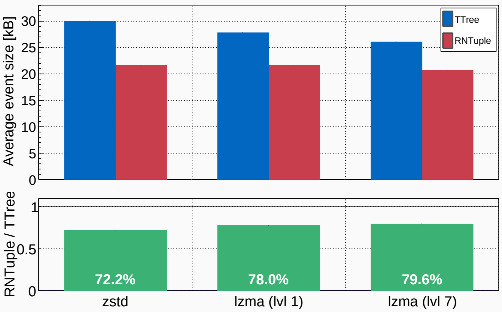
To answer these questions, we've evaluated DAOD_PHYS:

1. By creating RNTuple-based DAOD_PHYS files fully equivalent to their TTree counterparts...
   - Using a (sample of a) data set from real data and MC
   - Recompressing the original TTree-based samples with different algorithms using ROOT's `hadd`
   - Converting these samples to RNTuples using ROOT's `RNTupleImporter`
2. ...and comparing those in terms of storage efficiency and read throughput

Storage efficiency

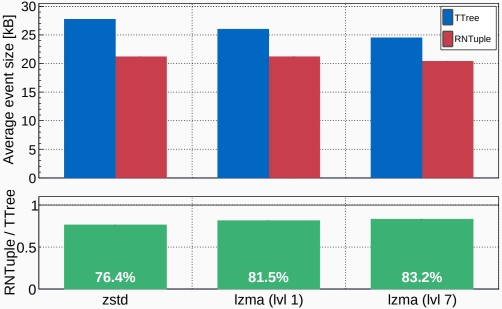


DAOD_PHYS storage efficiency, data



DAOD_PHYS storage efficiency, MC

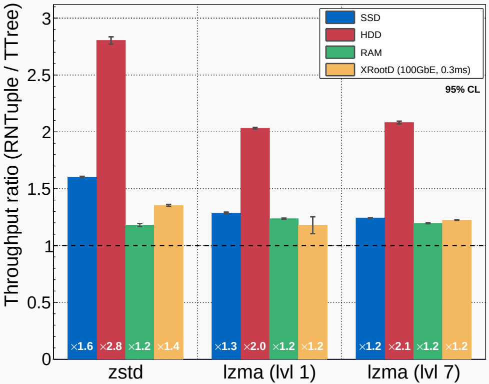
Storage efficiency



DAOD_PHYS storage efficiency, data

- Storage efficiency in line with other evaluations [previous talk]
  - DAOD_PHYS (almost) exclusively contains collections
- Potential optimization: using `std::vector<bool>` for selection flags instead of `std::vector<char>`

Read throughput



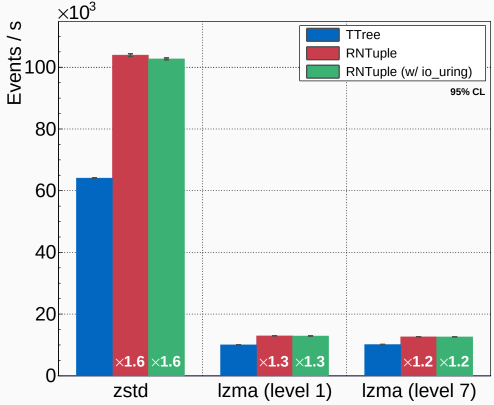
DAOD_PHYS RNTuple/TTree event throughput ratio, data

Comparable throughput for MC sample

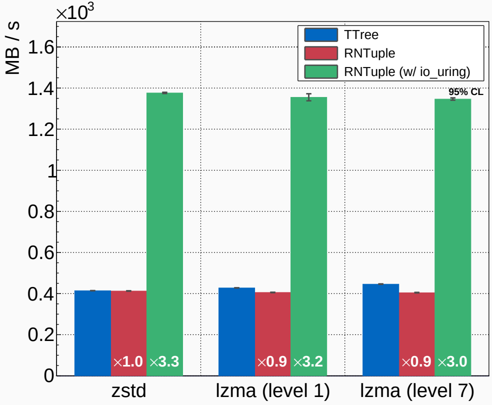
- Benchmarked with a (highly) artificial event loop using RDataFrame
  - Restricted to reading `std::vector<float>` columns
  - More representative benchmarks require additional RNTuple support in Athena
- Depending on storage medium, performance may become CPU-bound

Read throughput: SSD

io_uring: Linux interface for async I/O,utilized by RNTuple (requires Linux kernel version >= 5.1)



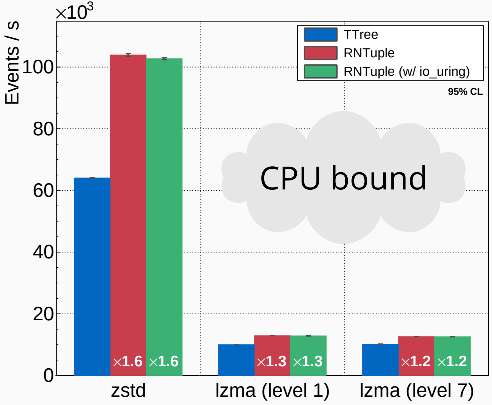
DAOD_PHYS RNTuple/TTree SSD event throughput



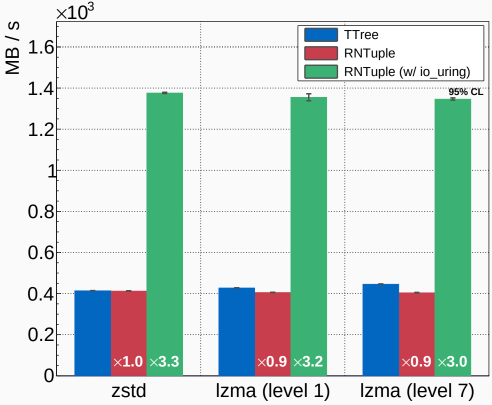
DAOD_PHYS RNTuple/TTree SSD raw I/O throughput

Read throughput: SSD

io_uring: Linux interface for async I/O,utilized by RNTuple (requires Linux kernel version >= 5.1)

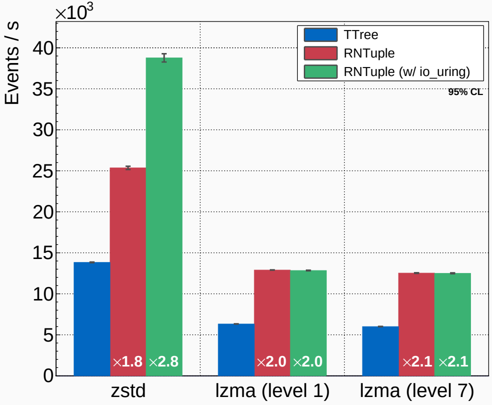


DAOD_PHYS RNTuple/TTree SSD event throughput

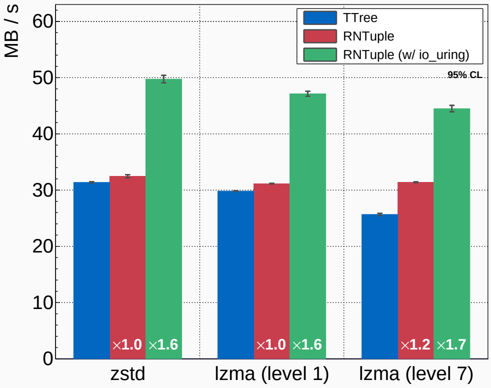


DAOD_PHYS RNTuple/TTree SSD raw I/O throughput

Read throughput: HDD

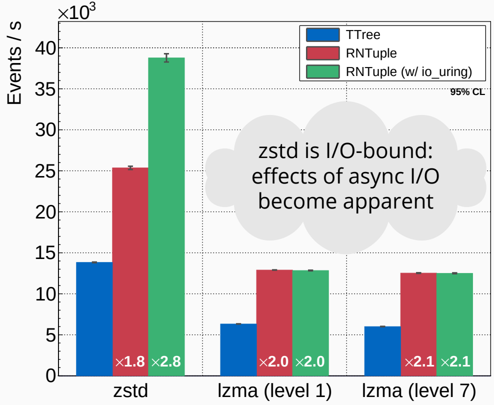


DAOD_PHYS RNTuple/TTree HDD event throughput

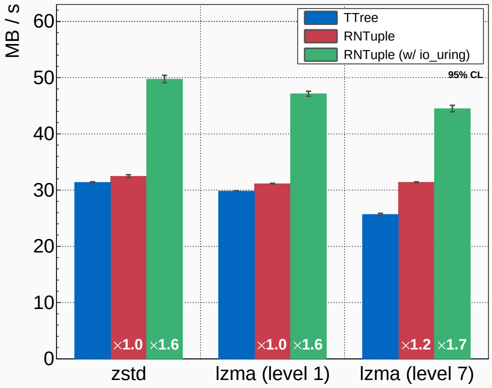


DAOD_PHYS RNTuple/TTree HDD raw I/O throughput

Read throughput: HDD



DAOD_PHYS RNTuple/TTree HDD event throughput



DAOD_PHYS RNTuple/TTree HDD raw I/O throughput

Next steps

1. Explore storage efficiency across more DAOD_PHYS data sets
2. Explore more of the evaluation phase space
   - Compression: lz4, lossy compression
   - Storage backends: Intel DAOS, S3 ▸ talk: “Storing LHC Data in DAOS and S3 through RNTuple”
   - Data sources: more (XRootD) latency configurations
   - RNTuple I/O parameters: page and cluster sizes ▸ poster: “ML-based Tuning of RNTuple I/O Parameters”
3. Benchmark with (multiple) representative analyses
4. Evaluate also with DAOD_PHYSLITE
5. Performing large-scale, distributed tests
6. Bonus: Evaluate RNTuple with other stages of the ATLAS data production pipeline

Summary and concluding remarks

- Support for RNTuple in ATLAS Athena is almost complete: validation ongoing
- RNTuple shows improvements in file size and read throughput w.r.t. TTree for DAOD_PHYS
- Similar to TTree, zstd outperforms lzma in terms of read throughput
  - File sizes are comparable, validation with more data sets is necessary
  - Comparison to other compression methods still planned
- We need further evaluation and benchmarking to understand current (performance) bottlenecks...

Summary and concluding remarks

- Support for RNTuple in ATLAS Athena is almost complete: validation ongoing
- RNTuple shows improvements in file size and read throughput w.r.t. TTree for DAOD_PHYS
- Similar to TTree, zstd outperforms lzma in terms of read throughput
  - File sizes are comparable, validation with more data sets is necessary
  - Comparison to other compression methods still planned
- We need further evaluation and benchmarking to understand current (performance) bottlenecks...

...but the benefits of using RNTuple for ATLAS are already apparent

Backup slides

Read throughput benchmark setup

- Single-core "analysis" using RDataFrame
- For 8 object containers, read the p_T, η, ϕ and m columns
  - Stored as `std::vector<float>`
  - Ergo, we read 32 branches (TTree) or top-level fields (RNTuple) in total
- Calculate the invariant mass (using `ROOT::VecOps`) and fill a histogram with the results
  - Done for every event, no cuts applied
- Each benchmark is repeated 10 times, outliers are removed

Hardware and software setup

CPU AMD EPYC 7702P @ 2GHz, 128 logical cores

RAM 128GB DDR4 RDIMM 3200 MHz

SSD Samsung MZWLJ3T8HBLS-00007

HDD TOSHIBA MG07ACA14TE SATA, 7200 RPM

Network 100 Gbit/s Ethernet

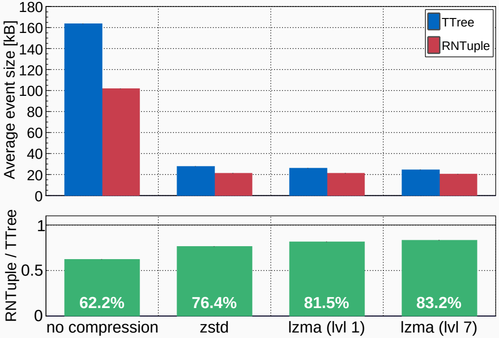
N.B. XRootD access from `projects.cern.ch` EOS instance (same datacenter)

ROOT GitHub

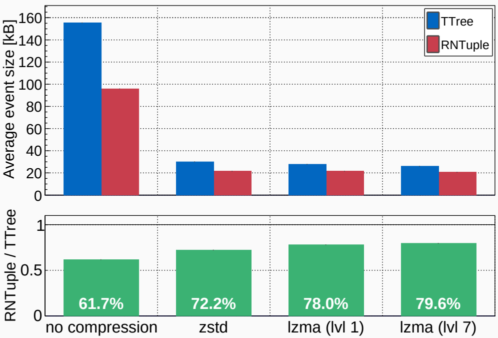
Benchmarks GitHub

OS AlmaLinux 9.1 with Linux kernel 6.3 from ELRepo (`uring` enabled)

Storage efficiency (with uncompressed)

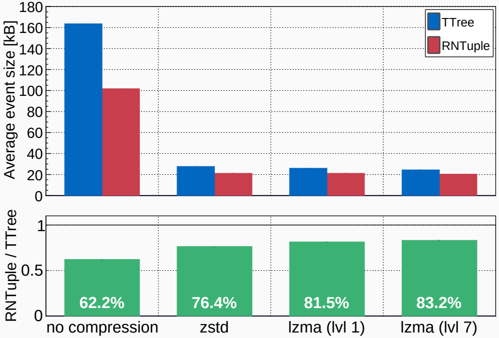


DAOD_PHYS storage efficiency, data



DAOD_PHYS storage efficiency, MC

Storage efficiency (with uncompressed)

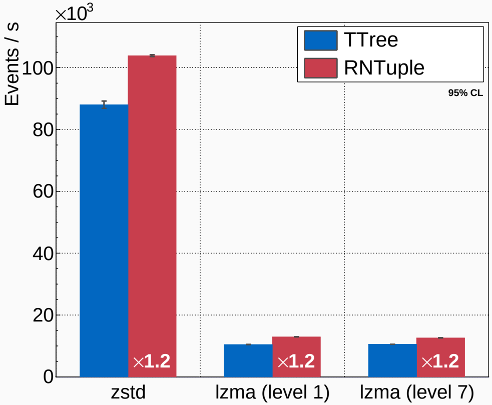


DAOD_PHYS storage efficiency, data

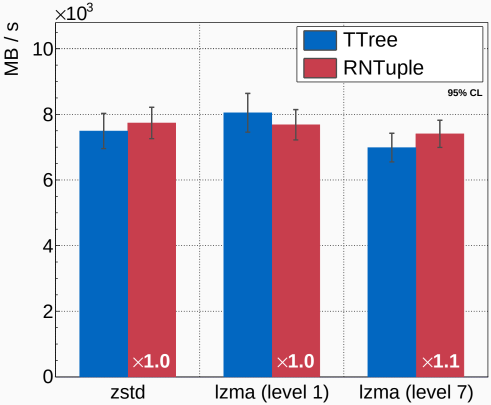
Why is the ratio RNTuple/TTree so much larger for uncompressed DAOD_PHYS?

- DAOD_PHYS files contain a lot of `std::vectors` and other collections
- Every `std::vector<POD>` needs 10 bytes more in TTree compared to RNTuple
  - Similar story for nested and other types of STL(-like) collections
- Lots of redundant data, compresses away well – but not completely

Read throughput: RAM



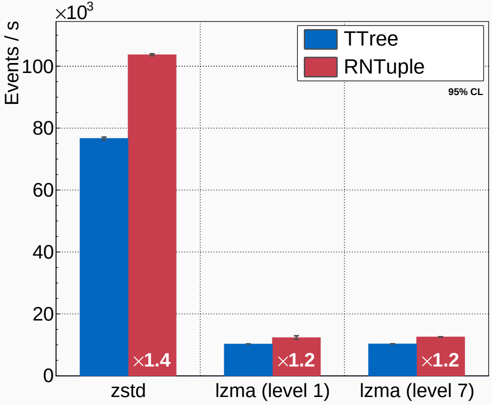
DAOD_PHYS RNTuple/TTree RAM event throughput



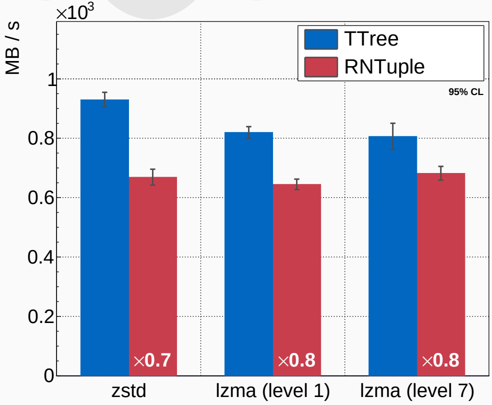
DAOD_PHYS RNTuple/TTree RAM raw I/O throughput

Read throughput: XRootD

RNTuple uses a different code path than TTree, XRootD access is not yet optimized for this



DAOD_PHYS RNTuple/TTree XRootD event throughput



DAOD_PHYS RNTuple/TTree XRootD raw I/O throughput