EPJ Web of Conferences **295**, 06020 (2024) https://doi.org/10.1051/epjconf/202429506020
CHEP 2023 FERMILAB-CONF-24-0690-CSAID

ROOT's RNTuple I/O Subsystem: The Path to Production

Jakob Blomer[1,*], *Philippe* Canal[2], *Florine* de Geus[1], *Jonas* Hahnfeld[1], *Axel* Naumann[1], *Javier* Lopez-Gomez[1], *Giovanna* Lazzari Miotto[1], and *Vincenzo Eduardo* Padulano[1]

[1]CERN
[2]Fermilab

Abstract. The RNTuple I/O subsystem is ROOT's future event data file format and access API. It is driven by the expected data volume increase at upcoming HEP experiments, e.g. at the HL-LHC, and recent opportunities in the storage hardware and software landscape such as NVMe drives and distributed object stores. RNTuple is a redesign of the TTree binary format and API and has shown to deliver substantially faster data throughput and better data compression both compared to TTree and to industry standard formats. In order to let HENP computing workflows benefit from RNTuple's superior performance, however, the I/O stack needs to connect efficiently to the rest of the ecosystem, from grid storage to (distributed) analysis frameworks to (multithreaded) experiment frameworks for reconstruction and ntuple derivation. With the RNTuple binary format soon arriving at its first production release, we present RNTuple's feature set, integration efforts, and its performance impact on the time-to-solution. We show the latest performance figures of RDataFrame analysis code of realistic complexity, comparing RNTuple and TTree as data sources. We discuss RNTuple's approach to functionality critical to the HENP I/O (such as multithreaded writes, fast data merging, schema evolution) and we provide an outlook on the road to its use in production.

1 Introduction

The ROOT RNTuple project is a multi-year R&D effort to re-design the TTree event data storage system for the HL-LHC era. [1–3] More than 1 EB of data of LHC Runs 1–3 are stored worldwide in the TTree format. The RNTuple format is expected to store in the order of 10 EB Run 4–6 data comprising all the experiment event data models (EDMs) stored in TTree today.

Developed in the context of ROOT 7 [4], RNTuple comes with a carefully updated event data binary layout and new C++ interfaces. RNTuple breaks backwards compatibility to TTree in order to fully exploit optimization opportunities on modern storage hardware and systems, such as SSDs, distributed object stores, and heterogeneous and highly parallel platforms. Our previous performance studies (summarized in Section 3) show that RNTuple compared to TTree produces significantly smaller files, often 15 % or more, and has significantly better throughput, often by factors from compressed data to histograms. Furthermore, RNTuple has shown to scale well on a small high-performance cluster served by a distributed object store [5], a technology that is hard to access with TTree. LHC experiments expressed

*e-mail: jblomer@cern.ch

interest to change the event data format to the RNTuple format within a timeline compatible with HL-LHC.

Excellent performance for typical analysis workflows is a key motivation for the RNTuple development. Yet, a useful I/O system for High Energy and Nuclear Physics has many more facets. Data needs to be efficiently written, so the I/O system needs to be integrated with potentially highly parallel experiment frameworks, and include means to efficiently derive data sets without full recompression ("fast merging and cloning"). With experiment life times of several decades, backward and forward compatibility are critical, both for the RNTuple data format itself and for the user-defined data models stored in RNTuple files ("schema evolution"). The I/O system furthermore needs to provide the flexibility to efficiently connect to current and future Grid and HPC storage systems, such as XRootD, high-performance cluster file systems or object stores.

This contribution summarizes the challenges and plans of moving the RNTuple prototype to ROOT's future production event I/O system. Figure 1 provides the timeline of the shift from TTree to RNTuple. We foresee to finalize the RNTuple binary format by the end of 2024. Afterwards, the RNTuple file format will be limited to backward-compatible (and to the possible extent forward-compatible) changes. We expect that the majority of TTree data will remain in that format; hence TTree will stay available in ROOT as a legacy option.

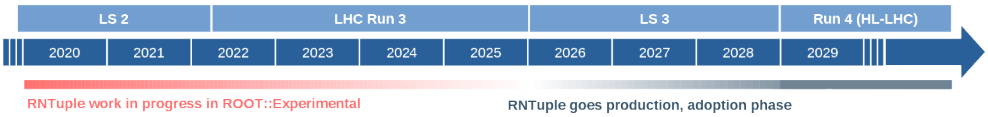

Figure 1. RNTuple release timeline. The version 1 binary format is expected by the end of 2024, followed by a ROOT 7 release that reads and writes the version 1 format. Future releases remain backward compatible.

2 State of the RNTuple prototype

The RNTuple system is developed within the ROOT sources [6]. It is part of the `root7` module. The classes are currently in the `ROOT::Experimental` namespace, i. e. they are still subject to changes. The RNTuple sources also comprise the format specification, which has been used already by third parties to build readers.

In its current state, RNTuple can read and write data with an API similar to the TTree one, including bulk reading. RDataFrame processes RNTuple and TTree data alike. RNTuple's supported type system is more limited than TTree's, yet already powerful enough to represent, for instance, CMS nanoAODs, and ATLAS PHYS, PHYSLITE, and full AOD files. The ATLAS and CMS frameworks have included experimental RNTuple models in their integration builds.

3 Latest performance results

In this section, we show the performance of RNTuple on several small but realistic analysis benchmarks (see Table 1). Figure 2 compares the storage efficiency of the benchmark input data for TTree and RNTuple. The main contributor of RNTuple's space savings is a more compact representation of collections and boolean values. Furthermore, RNTuple uses type-dependent data encodings (such as byte shuffling, delta encoding, zigzag encoding) that optimize certain columns for better compression ratio.

Table 1. Sample analyses for performance evaluation. The first three benchmarks have been introduced earlier. [2]

LHCb run 1 open data B2HHH	H1 micro dst [×10]	CMS nanoAOD June 2019	ATLAS OpenData
18/26 fields (>75 %)	16/152 fields (~10 %)	6/1479 fields (<1 %)	13/81 fields (~15 %)
fully flat data model	event sub collections	event sub collections	`std::vectors`
8.5 million events	2.8 million events	1.6 million events	7.8 million events
24 k selected events	75 k selected events	141 k selected events	76 k selected events

Figure 3 compares the throughput of an RDataFrame implementation of the benchmarks, with RNTuple and TTree input data, respectively. The benchmarks compare single-core performance, input data is zstd compressed and comes from different sources. In no case RNTuple is slower than TTree, typically it is substantially faster. The main contributors to higher throughput are the smaller input data size to start with, asynchronous reading by default in RNTuple, parallel I/O to SSDs, and fewer instructions in the hot I/O code path. Note that our previous performance comparisons used hand-optimized event loops for TTree and RNTuple. The RDataFrame based comparison is closer to real-world analysis use cases.

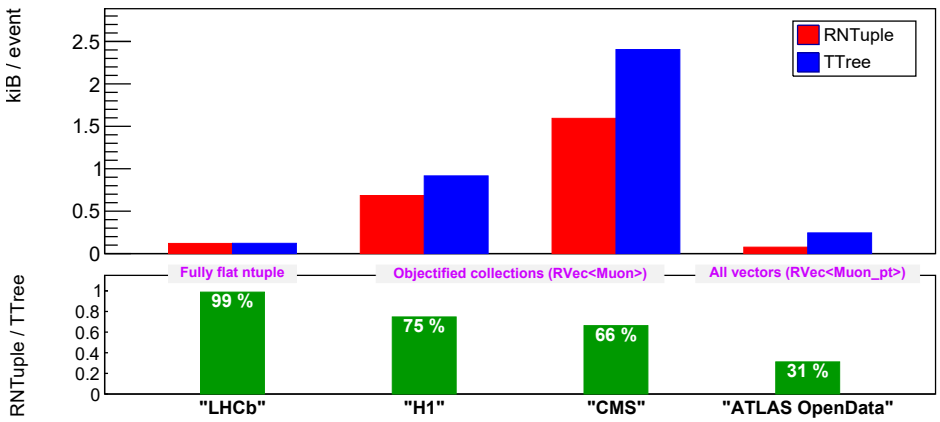

Figure 2. Comparison between TTree and RNTuple for the average event size of the RNTuple standard benchmarks. Input data is compressed with zstd.

Previous work compared RNTuple to HDF5 and Parquet [7] showing a clear performance advantage of RNTuple. It's noteworthy that the HDF5 results may vary depending on the effort put into adapting its inherent tensor layout to the columnar access pattern predominant in HENP analyses. Furthermore, RNTuple has shown to scale on a >100 cores HPC cluster to an aggregated throughput of more than 35 GB/s and more than 500MB/s single core throughput [8]. Input data in this benchmark was a version of the "LHCb" benchmark scaled-up to 1 TB, served from a DAOS HPC object store. This benchmark validates basic scalability of analysis code using distributed RDataFrame on RNTuple data sets.

4 Type system

High Energy Physics tends to use a rich type system in the experiment data models. Even the simplest tasks typically require at least vectors of simple data types, i. e. a more complex

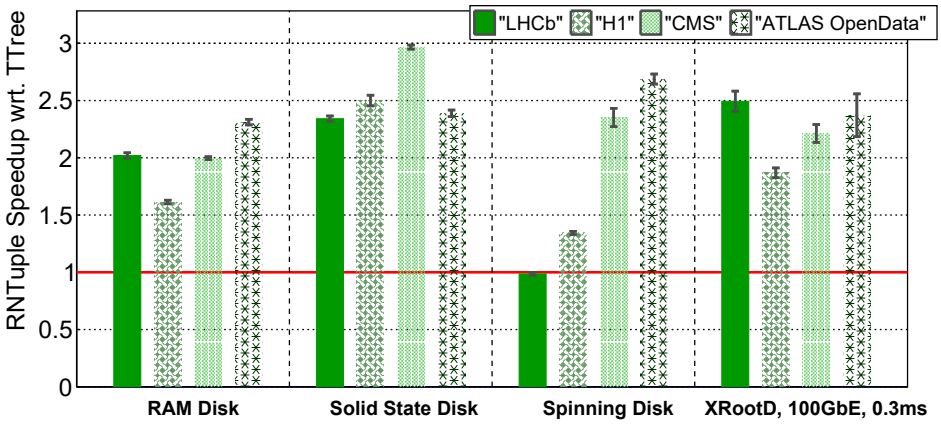

Figure 3. Throughput comparison between RNTuple and TTree for the RDataFrame implementation of the RNTuple standard benchmarks, single threaded. Input data comes from different sources and is zstd compressed. Benchmark environment is Alma Linux 9 with uring support (5.x mainline kernel), AMD EPYC 7702P, 128 GB RAM, 100 GbE.

Table 2. Types that can be persistified with RNTuple. Precision cascades were recently introduced as a mechanism to separate several levels of lossy compression into different files. [9]

PoD	`bool, (unsigned) char, std::byte, (u)int[8,16,32,64]_t, float, double`	Flat n-tuple	Reduced AOD	Full AOD / ESD / RECO	Available
(Nested) vectors	`std::vector, RVec, std::array,` C-style fixed-size arrays				Available
String	`std::string`				Available
User-defined classes	Non-cyclic classes with dictionaries				Available
User-defined enums	Scoped / unscoped enums with dictionaries				Available
User-defined collections	Non-associative collection proxy				Available
`stdlib` types	`std::atomic, std::pair, std::tuple, std::bitset, std::(unordered)set, std::(unordered)map`				Available
Alternating types	`std::variant, std::unique_ptr, std::optional`				Available
Intra-event links	`"&Electrons[7]"`				In design
Low-precision floating points	`Double32_t, Float16_t, (b)float16`	*Optimization benefitting all EDMs*			Available
	Custom precision and range				In design
	Precision cascades				In design

data representation than a flat table. Higher-level data formats that are used for simulation and reconstruction by the frameworks typically persistify a broad set of C++ types. Table 2 summarizes the type support in RNTuple.

User-defined classes and enumerations require ROOT dictionaries, i. e. cling-provided type reflection information. Conversely, simple and `stdlib` types in RNTuple can be serialized without dictionaries. User-defined collections are supported through ROOT collection proxies, identical to their support in TTree.

The TTree type system is more powerful still. For instance, it can store arbitrary object relationships where objects hold (raw) pointers to each other and it supports dynamic polymorphism, i. e. runtime type discovery when serializing a pointer to a base class. Those

two features will not be supported in RNTuple, following discussion with large experiments. These features turned out to be both costly in terms of code maintenance and negatively impacting overall performance.

While the columnar layout is desired for analysis workflows, reconstruction workflows may require a row-wise data layout. During reconstruction, typically all the data members of an event are read. Due to the larger event sizes, it is more difficult to fill the column buffers given typical memory constraints. RNTuple support for row-wise storage is ongoing work. It will be based on the support for BLOB fields, which have an on-disk layout comparable to vectors of bytes.

5 Writing data and data derivation

In this section, we discuss the required mechanisms to create RNTuple data sets. RNTuple provides an API to serialize C++ in-memory objects to disk entry-by-entry, similar to TTree. RNTuple data generation through `RDataFrame::Snapshot()` is ongoing work.

Entry-by-entry writing becomes challenging in highly parallel environemnts. RNTuple allows multi-threaded frameworks to prepare entries independently of each other and it can leverage the available cores for the parallel compression of data pages. Still, object serialization and writing into a single file are synchronization points that impact scalability to high core counts. Ongoing work implements a more scalable way of multi-threaded writing. The RNTuple binary layout groups entry ranges in self-contained, relocatable clusters of the order of 100 MB. Threads can, in principle, independently of each other prepare and write to a single file entire clusters. In this write mode, synchronization points between cluster commits from different threads will be very short and compatible with spin locks.

A particular challenge posed by the experiment frameworks is the need to extend the data schema after the write process started. TTree addresses this challenge through "backfilling", i. e. the ability to add a new branch at any point and to set its values for the previous entries. RNTuple provides a slimmed down version of this feature, called "late schema extension", that allows for adding deferred top-level data fields that are zero-initialized for the initial entries.

5.1 Fast data re-shaping

An important mechanism in HENP data management is the ability to efficiently derive new data sets from existing data without reprocessing and recompressing the entries. The ROOT I/O provides "fast merging" (vertical merge of data sets with compatible schemas) and "fast cloning" (discarding columns from existing data sets). Fast merging and fast cloning only update meta-data and write the new data as carbon copy of existing blocks. This mechanism has been successfully prototyped in RNTuple and is currently being implemented as production code. The RNTuple format is prepared to implement horizontal merges ("materialized friends"), which is a potential extension after the first production release.

An interesting variant is "zero copy merging and cloning" that leverages the copy-on-write capability of modern file systems. In this mode, the physical data copy operation can be avoided. RNTuple has shown a proof-of-concept exploitation of file system block sharing support. [10]

5.2 Data joins

High-Energy Physics frequently uses a limited set of data joins: union of data sets ("chains") and 1:1 or 1:n joins ("friends", possibly indexed). Support for these joins is already partially

available and planned to be complete for the first production release. An R&D program in approval on more advanced use cases, such as stored filters (e.g., event bitmasks), indexed joins, and provenance meta-data, is considered a potential extension after the first production release

6 Format transition and compatibility

While the RNTuple binary format and the native API breaks backwards compatibility to TTree, RNTuple aims at a smooth integration with the well-established ROOT/HEP ecosystem. RNTuple data and TTree data can co-exist in (the same) ROOT files. RNTuple provides automatic conversion of TTree data into RNTuple data (`RNTupleImporter` class).

RNTuple integrates with the commonly used ROOT I/O tooling, such as the browser and the `hadd` utility[1]. Integration with the `TBufferMerger` and `TMPIMerger` parallel and distributed data merger infrastructure is planned. RNTuple will also adopt the TTree automatic schema evolution rules as well as support for the infrastructure around the I/O customization rules ("read rules"). Support for read rules affecting only transient data members is already available. Full schema evolution support for user-defined classes is planned for the first production release.

Analysis code written for RDataFrame needs no changes when the input data changes from TTree to RNTuple. Direct users of the TTree API, such as experiment frameworks, need to adapt to the new RNTuple interfaces. This is an intentional change. The new interfaces resemble the concepts established in TTree (such as filling and loading entries) but for robustness they follow the modern C++ core guidelines.

There is no plan to implement the `TTree::Draw()` interface in RNTuple. However, there is ongoing discussion on a ROOT plotting interface based on RDataFrame with a similar level of expressiveness and conciseness.

7 Adoption

The RNTuple prototype, as available in the ROOT head of development, has been successfully integrated as experimental options in both the CMS and ATLAS software frameworks. In CMSSW, a nanoAOD output module is able to write nanoAOD files in the RNTuple format. Athena can read and write ATLAS full AOD and derived AOD files in the RNTuple format. These early adoption efforts provided very valuable feedback to the RNTuple development, for instance regarding the required type support and modes of writing.

The uproot software [11] has re-implemented reading of RNTuple data, based on the available RNTuple specification. In the future, we plan to provide as part of ROOT a C interface that will give access to the RNTuple low-level functionality (e. g. schema information, reading of columns). A C interface should significantly lower the barrier of providing RNTuple functionality in languages other than C++ and Python, such as Julia or Rust.

8 Summary

In this contribution, we have shown the path for moving the ROOT RNTuple I/O system from an R&D effort to a production event data I/O system for High-Energy and Nuclear Physics. Besides fast reading of ragged, columnar data, a production I/O system for HENP experiments needs to provide an additional, rich feature set catering both to end users and

[1] `hadd` integration to be completed at the time of writing

experiment frameworks. The additional feature set include automatic and manual schema evolution for user-provided data models, mechanism for efficient writes in highly parallel environments, fast merging and cloning for the efficient construction of derived data sets, horizontal and vertical data joins, and support for a comprehensive type system covering a substantial subset of the types available in C++. These features are difficult to find in off-the-shelf I/O systems from industry or the HPC world.

At the same time, the RNTuple development aims at reducing the feature set for event data compared to current production ROOT I/O in order to avoid, on the path forward, promoting functionality with a severe performance impact or that becomes disproportionally difficult to maintain. There is ongoing collaboration with large experiments on the final feature set of the first production release, the timeline, the transition path, and the validation of RNTuple at scale. These efforts are on track for mounting RNTuple as a production system for HL-LHC data.

References

[1] R. Brun, F. Rademakers, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment A **389**, 81–86 (1997)
[2] J. Blomer, P. Canal, A. Naumann, D. Piparo, EPJ Web Conf **245**, 02030 (2020)
[3] The ROOT Team, Tech. Rep. FERMILAB-FN-1165-SCD, Fermilab (2021)
[4] *ROOT 7*, https://root.cern/for_developers/root7/, accessed: 2023-11-27
[5] J. Lopez-Gomez, J. Blomer, EPJ Web Conf **251** (2021)
[6] The ROOT Team, *ROOT*, https://github.com/root-project/root (2023)
[7] J. Lopez-Gomez, J. Blomer, Journal of Physics: Conference Series **2438** (2023)
[8] V.E. Padulano, E.T. Saavedra, P. Alonso-Jordà, J.L. Gómez, J. Blomer, Cluster Computing **25** (2022)
[9] Y. Ying, *Precision Cascade: A novel algorithm for multi-precision extreme compression*, in *Proc. of 21st International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT)* (2022)
[10] E. Marinelli, *Zero-copy merge with RNTuples*, https://cds.cern.ch/record/2834291 (2022)
[11] J. Pivarski, P. Das, C. Burr, D. Smirnov, M. Feickert, T. Gal, L. Kreczko, N. Smith, N. Biederbeck, O. Shadura et al., *scikit-hep/uproot3: 3.14.4* (2021), `https://doi.org/10.5281/zenodo.4537826`