Adoption of ROOT RNTuple for the next main event data storage technology in the ATLAS production framework Athena

Marcin Nowak[1,*] *Peter* van Gemmeren[2]*Alaettin Serhan* Mete[2]and *Tatiana* Ovsiannikova[3]
on behalf of the ATLAS Computing Activity

[1]Brookhaven National Laboratory (US)
[2]Argonne National Laboratory (US)
[3]University of Washington (US)

Abstract. Since the start of LHC in 2008, the ATLAS experiment has relied on ROOT to provide storage technology for all its processed event data. Internally, ROOT files are organized around TTree structures that are capable of storing complex C++ objects. The capabilities of TTree developed over the years and are now offering support for advanced concepts like polymorphism, schema evolution and user defined collections and ATLAS makes use of these features to handle its Event Data Model (EDM). But some original TTree concepts, like the POSIX file model and sequential writing, remain unchanged since the beginning and could be an obstacle to achieving the performance required for High Luminosity LHC.
With the HL-LHC performance goals in mind, the ROOT project developed a new storage format - the RNTuple. RNTuple, with its accompanying user API, is now in the final development stage and is planned to be production-ready at the end of 2024. Soon after that, the TTree will become a legacy format. ATLAS intends to have its main event processing framework Athena ready to use RNTuple in the production environment as early as possible. The work on adopting RNTuple as another ROOT storage technology in Athena started already in 2021 and is now nearly complete. Although the initial goal was to focus only on derived-AOD products (PHYS and PHYSLITE), with a little added effort all ROOT-based data products of ATLAS can be now stored in RNTuple format and transparently read back.
In this paper we will describe the current state of RNTuple adoption in the Athena framework and explain the ATLAS EDM requirements that had to be met on the ROOT side to successfully integrate both environments. We will demonstrate the ability to run standard ATLAS production workflows, based on RNTuple as the event data storage technology, and point out key advantages of the new format.

1 Introduction

The ATLAS experiment [1] is the biggest of the four particle detector experiments at LHC, CERN. ATLAS was formed in 1992 and has been collecting data since 2009, when the LHC

*e-mail: mnowak@bnl.gov

Run 1 started, up to the currently ongoing LHC Run 3. The collected event data is processed by Athena - the official, universal software framework that has been in development and use since 2002. The software complexity reflects that of the detector - in terms of lines of source code, Athena consists of 4 million lines of C++ and 1 million lines of Python. The framework depends on many external projects, particularly ROOT for data persistency. The Event Data Model (EDM) used by Athena contains hundreds of C++ classes, most of which are stored in files. ATLAS is using ROOT TTree as the storage format and utilizes many advanced TTree I/O features required to handle its EDM.

During many years of data taking, Athena evolved from a single process executable into a multithreading and multiprocessing capable application. These changes were necessary to improve throughput through concurrent processing or to reduce overall memory usage. In this paper we describe yet another, very significant evolution Athena is undergoing right now - the migration from ROOT TTree to RNTuple [3] - which is an important step in preparation for the massive data rate increase planned for LHC's Run 4, called the High Luminosity LHC.

2 Rationale behind the migration to RNTuple

Handling High Luminosity LHC data rates and data volumes will be a challenge. Assuming sustained (flat) year-on-year storage hardware budget increases, even foreseeing improvements in new hardware, the predictions (Figure 1) show that storing event data with the current efficiency would suffer from storage shortage.

Increasing the compression strength (level) would be a tempting choice in this situation. However, ATLAS is already using the strongest compression that is considered acceptable (LZMA for all upstream data) and additional gains in storage are limited, typically to around 10%. On the other hand, migration to RNTuple brings the possibility to achieve similar or even better storage space savings without incurring an extra CPU load. As shown in Sect. 6, this is actually the case - Athena data files with real data (or MC-generated ones) use up to 20% to 40% less space than their TTree-based equivalents, filled with exactly the same data.

Space savings are not the only improvement we expect to gain from migrating to RNTuple. As a recently developed modern software, RNTuple was designed to take advantage of I/O related features that did not exist before: asynchronous I/O, SSD-optimized access patterns, zero-copy, and direct object store interfacing. While not yet tested by ATLAS, these features should help in handling the increased HL-LHC data rates.

Last but not least, the ROOT team already announced that TTree will become a legacy format when RNTuple enters the production stage, that is now expected to happen at the beginning of 2026. That means that it will have priority in getting updates and improvements, some of which may even remain exclusive to it. It is also possible that some future ROOT tools, e.g. for data analysis, may be developed for RNTuple only.

With all the above reasons in mind, ATLAS began investigating RNTuple for Athena already in 2021 and is already able to use it for all its data processing stages.

3 Migration stages

The migration of Athena to RNTuple took three years and can be divided into several phases. In the beginning, we focused solely on the data model and API compatibility. The ideal solution was for the new storage technology to come as a new persistency plug-in. (Athena, implementing concepts coming from the Gaudi [5] framework, supports interchangeable, dynamically loaded technology-specific plugins). To create a new, fully interchangeable plugin that would be able to replace the existing TTree-based one, RNTuple needed to handle

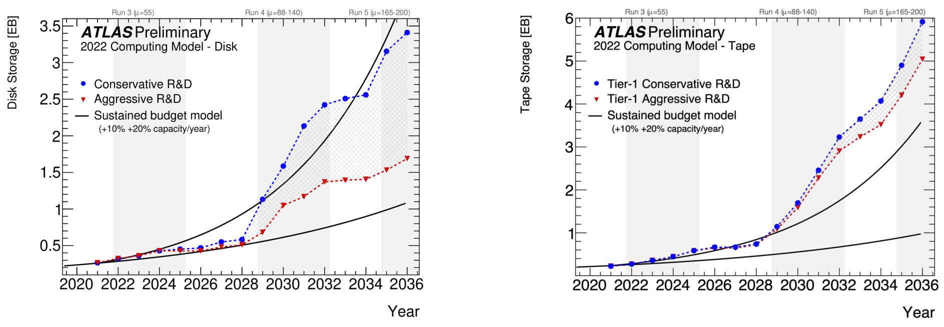

Figure 1: Projected evolution of disk and tape usage under the conservative and aggressive R&D scenarios. The black lines indicate the impact of sustained year-on-year budget increases, and improvements in new hardware, which together amount to a capacity increase of 10% (lower line) and 20% (upper line). The vertical shaded bands indicate periods during which ATLAS will be taking data. Source: ATLAS HL-LHC Roadmap [4]

all existing Athena persistent types and to work with the object manager (StoreGate) that controls all Athena in-memory objects. Once that was achieved, we could focus on the operational side - running multithreaded and multiprocess production jobs and measuring performance. We also needed to teach the standalone Python tools that automatically configure and later validate jobs to extract data from RNTuple files.

The migration is not officially completed yet - one of the workflows is still in development (see Sect. 5), there are larger scale data challenges planned for 2025, and the RNTuple itself will not enter production until 2026. However, the most important milestones have already been reached.

The migration process so far, divided into stages, is described in detail in the following sections.

3.1 Data Model Compatibility

The Event Data Model (EDM) is a set of types (or classes) used to represent ATLAS event information in application memory. In activities involving data storage, this EDM is often called the transient EDM to distinguish it from the persistent EDM, which consists of types used to store event data on persistent media. At the beginning of ATLAS and LHC operation, these two sets were mostly disjoint to avoid exposing complicated (and, at the time, unsupported) C++ constructs to the persistency layer. Now, at LHC Run 3, thanks to the many improvements in the ROOT/TTree capabilities, the two EDMs are almost overlapping. However, this means that RNTuple was tasked with handling a complicated ATLAS EDM from the very start.

One of the first steps of the migration was to take inventory of the current persistent EDM and share with the ROOT development team the list of required features that needed to be supported if the RNTuple was to be able to successfully replace TTree. This list contained the following requirements, which are now implemented and ready to use:

3.1.1 STL containers

ATLAS uses std::vectors extensively, both in the transient and in the persistent EDM. RNTuple was supporting std::vectors very early on - this particular requirement was added simply to reinforce the fact that it was used so widely, often with a nesting depth of three levels. Singular uses of std::map and std::set were discovered in some of the data formats - they are nowadays also supported by RNTuple.

3.1.2 Read rules

Read rules are small fragments of user provided C++ code that are executed every time an object of a particular type is loaded for a new event. This code is not part of the regular application C++ sources, but is provided together with the XML instructions for ROOT class dictionary creation and is compiled into the actual dictionary library. The basic use of read rules is to call a method that will bring the object to the correct state after reading. This feature is for example used in Athena to reinitialize smart pointers. A more advanced scenario is to directly modify object data members after reading - an approach used to deal with EDM changes introduced by schema evolution.

3.1.3 User-defined collections

Support for user-defined collections in ROOT persistency is realized through *collection proxies*. A collection proxy offers a unified interface for collection access and modification, independent of the actual implementation. Athena offers proxies for all its data containers (xAOD Containers [6]) which are internally std::vectors of pointers (DataVectors). RNTuple does not support pointers, but can access individual vector elements through the proxy interface without knowing anything about them. Collection proxies were originally supported for the TTree I/O and can be now also used with RNTuples.

3.1.4 Dynamic attributes

Dynamic attributes in the ATLAS EDM are optional data members that are not part of the fixed class definition (in the C++ sense). They can be added to an xAOD container definition at any point of a job execution, even when many events with that container were already written out. Because they are listed as class members, they can not be part of a precompiled class dictionary and are not handled by ROOT I/O operations for that particular container type. However, thanks to the Dynamic Model Extensions API, an RNTuple model can be extended at any time when writing, to add a new top-level column for storing a particular dynamic attribute. Entries in that column prior to the first attribute occurrence are considered to be empty.

4 API Compatibility

The current Athena I/O layer was designed many years ago with the primary goal of making use of the contemporary TTree API, resulting in a predominant use of void* pointers and information about the actual type carried by the TClass instances. The void* type is used to pass arguments across many layers of Athena, making it very hard to change at this point. Object ownership is usually exclusive, with the Athena internal object event store (StoreGate) managing all transient objects, and performing store cleanup after each event. All objects with event data are thus recreated for every event. When reading from storage, Athena entrusts object creation to the ROOT I/O layer and assumes ownership of these objects without copying them.

By default, RNTuple provides a modern, templated API for object reading and delivers objects through std::shared_ptr. However, converting a shared_ptr into a regular C pointer makes controlling the lifetime of the object problematic. Additionally, using a templated API in a framework's generic I/O layer is not feasible. For these reasons - and to avoid changes in Athena components that are still used for TTree I/O - ATLAS requested the implementation and maintenance of the "old style" typeless API also for RNTuple, including support for passing exclusive object ownership to Athena.

Considering the various ATLAS EDM requirements mentioned before, in terms of API adaptations, we found out that little changes were necessary on either side. Read rules and collection proxies are not a part of a specific TTree or RNTuple API, but rather a separate I/O concept. The adoption of these features for RNTuple as-is allowed us to keep the API the same. Dynamic model extending is part of the new RNTuple API, but handling dynamic attributes is a late addition to Athena and implemented deep enough in the I/O layer to be a part of the actual technology specific plugin. It should be noted that ATLAS intends to keep the current Athena I/O API for both TTree and RNTuple, because of the need to be able to read both formats for the lifetime of the experiment.

5 Operational Compatibility

Athena originated as a single process application, but for performance reasons evolved - first into a multiprocess (AthenaMP) and later (Run 3) even to a multithreaded (AthenaMT) framework. AthenaMT did not completely replace AthenaMP for two reasons: limited scalability with the number of threads of the MT version and the inability of some algorithms in data analysis to run in an MT environment. Therefore, both versions are used by ATLAS in the production environment today. In particular, the DAOD production runs exclusively in the MP mode.

AthenaMP's biggest performance challenge is an efficient merging of the outputs from the individual worker processes into a single output file. To avoid merging output files in a (rather slow) separate, single process step following event processing, an in-memory merging component was developed - the SharedWriter. SharedWriter can currently operate in two modes:

Legacy mode - with objects sent to the writer one by one and all TFile writing done by the writer process

Parallel mode - with all workers independently writing to their local TMemFiles and transferring it incrementally to the writer to be merged by ROOT's TFileMerger

The Parallel mode scales better with the number of workers, as compression is performed in parallel by the workers, spreading the CPU load. On the other hand, the Legacy mode uses less memory, as writing buffers are allocated only by the SharedWriter. Also, it has been observed that for TTrees, the Legacy mode produces slightly more optimized files, as there is only a single cluster being written during the writer's learning phase (whereas in Parallel mode, each worker produces an unoptimized first cluster). Currently, ATLAS can only use the Legacy mode with RNTuple, because the RNTuple merging functionality required for the Parallel mode has not yet been implemented.

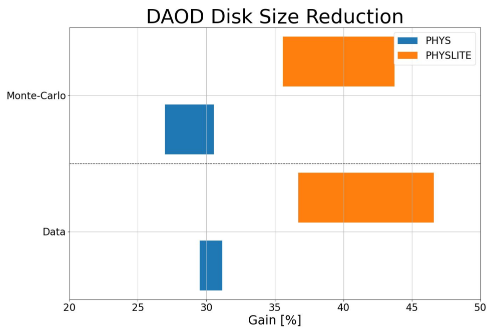

Figure 2. Space savings achieved by changing storage technology from TTree to RNTuple, shown as percentages of the original file size. The gains depend on the derived-AOD product (PHYS/PHYSLITE) and the data type (data/MC). The width of the bars represents variations for data coming from different data taking years. Source: [7]

6 Performance

One of the most anticipated advantages of migrating to RNTuple is the reduction of file sizes. After achieving a fully functional RNTuple version of Athena, we were able to compare output file sizes of identical production jobs, using actual ATLAS data as input and producing identical outputs, except with either TTree or RNTuple formats. We observed a reduction in size for almost every type of data object, except for some specific data fields that contained nested vectors of integer data (i.e. std::vector<std::vector<int»). We also observed, as shown in Figure 2 that more derived data formats compress better: data PHYSLITE shows over 40% size reduction.

It is now understood that the new method used by RNTuple to serialize arrays of numerical data (byte-shuffling) may, in some cases, disrupt patterns in original data representation that previously allowed for better compression. To prevent this, an option was added to RNTuple to allow selecting the serialization method for each individual column.

No particular difference in speed was observed for typical production jobs. That in itself is not very surprising, as the tests were performed with the typical POSIX I/O, regular TFiles and the same storage hardware. Eventual improvements may result from using new advanced I/O features later on.

Memory measurements of the test jobs showed a somewhat higher memory usage than with TTrees (<10% increase). Not much tuning has been done in that direction yet. The important goal remains - to fit into the 2GB per process limit for Grid jobs. More testing in that direction is foreseen in 2025.

7 Non-Athena Access

In the normal production flow, the output files produced by Athena are later used by three types of readers: Athena itself (in case there are remaining processing steps to execute), Python standalone tools (to access in-file metadata or to perform output validation), and, most importantly, non-Athena analysis jobs, which are used to perform further analysis of the DAOD content.

During the Athena migration to RNTuple, we naturally implemented both writing and reading. The reading is fully transparent, allowing any standard Athena job can read files in both formats. Python standalone tools had to be migrated independently, as they often internally used hardcoded, direct access to TTrees. These Python tools are an integral part of the Athena ecosystem - they run as part of the job configuration to set job parameters based on the input file content and perform output file validation once the job finishes.

All important Python tools were migrated to RNTuple. The migration was greatly aided by good RNTuple Python bindings.

The analysis level access to Athena RNTuple files is currently in development - independently, but in close collaboration with the work done in Athena.

8 Recap and Summary

The Athena interface to RNTuple was implemented as a technology-specific plugin library, and is now one of the three coexisting technologies supported by ROOT: RNTuple, TTree and TKey storage. The specifics of each technology are entirely encapsulated by the plugins, which provide identical interfaces and the same functionality to the higher layers of the Athena framework. The choice of technology used for the output is, for convenience, controlled by a single job configuration flag. On the technical level, Athena is capable of producing multiple output streams with different technologies at the same time. For physics analysis, Athena can write all officially supported data products (RDO, HITS, ESD, AOD, DAOD) into RNTuple format.

Reading RNTuple (and TTree) files in Athena is completely transparent, without prior knowledge of the format of the input files. The same job can read inputs with different formats and even navigate between them.

This flexibility and transparency are possible because the RNTuple support for all Athena requirements concerning EDM and API allows using a single Athena build for both formats. In fact, runtime tests with RNTuple are now part of the continuous integration and nightly tests of the main branch of Athena.

The migration to RNTuple took significant time and work, both from ATLAS and ROOT [9] developers. Some fine-tuning is still needed, but we are nevertheless satisfied with the results achieved so far. We have seen very exciting results in file size reduction, and we are planning large scale tests to fully validate RNTuple for production.

Brookhaven National Laboratory's work was supported by the U.S. Department of Energy's Office of Science, Office of High Energy Physics, under Contract DE-SC0012704.
Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

References

[1] ATLAS Collaboration, (2008) The ATLAS Experiment at the CERN Large Hadron Collider. J. Inst. 3, S08003 DOI: 10.1088/1748-0221/3/08/S08003

[2] R. Brun, F. Rademakers, (1996) ROOT: An Object Oriented Data Analysis Framework. AIHENP'96 Workshop Nucl. Inst. & Meth. In Phys. Res. A 389 81-86. http://root.cern.ch

[3] J. Blomer, P. Canal et al, (2024) ROOT's RNTuple I/O Subsystem: The Path to Production. EPJ Web of Conf. **295** 06020. https://doi.org/10.1051/epjconf/202429506020

[4] Z. Marshall, A. Di Girolamo, (2022) ATLAS Software and Computing HL-LHC Roadmap, LHCC-G-182, CERN-LHCC-2022-005

[5] G. Barrand et al, (2001) GAUDI — A software architecture and framework for building HEP data processing applications, Computer Physics Communications, Vol. 140, Issues 1–2, https://doi.org/10.1016/S0010-4655(01)00254-5

[6] T. Eifert, M. Elsing, D. Gillberg, K. Koeneke, A. Krasznahorkay, E. Moyse, M. Nowak, S. Snyder, P. Van Gemmeren, (2015) Implementation of the ATLAS Run 2 event data model. Proceedings of the 21st International Conference on Computing in High Energy and Nuclear Physics J. Phys.: Conf. Ser. DOI: 10.1088/1742-6596/664/7/072045

[7] T. Ovsiannikova, P. Van Gemmeren, A.S Mete, M. Nowak, (2024) Impact of RNTuple on Storage Resources for ATLAS Production (in these proceedings)

[8] A.S Mete, M. Nowak, P. Van Gemmeren, (2024), Persistifying the complex event data model of the ATLAS Experiment in RNTuple. ACAT Conference proceedings, ATL-SOFT-PROC-2024-002 https://cds.cern.ch/record/2905189

[9] F. de Geus, J. López-Gómez et al, (2024), Integration of RNTuple in ATLAS Athena. EPJ Web of Conf. **295** 06013 https://doi.org/10.1051/epjconf/202429506013