ATLAS software tools to handle ROOT RNTuple*

Grigori Rybkin[1,**], *Alaettin Serhan* Mete[2], *Marcin* Nowak[3], and *Peter* Van Gemmeren[2]
on behalf of the ATLAS Computing Activity

[1]Université Paris-Saclay, CNRS/IN2P3, IJCLab (FR)
[2]Argonne National Laboratory (US)
[3]Brookhaven National Laboratory (US)

Abstract. The software of the ATLAS experiment at the CERN LHC accelerator contains a number of tools to inspect (validate, summarize, peek into etc.) all its official data formats recorded in ROOT files. These tools — mainly written in the Python programming language — handle the ROOT TTree which is currently the main storage object format of ROOT files. However, the ROOT project has developed a successor to TTree, called RNTuple. The new storage format offers significant improvements and ATLAS plans to adopt it in LHC Run 4. Work is ongoing to enhance the tools in order to handle the RNTuple storage format in addition to TTree in a transparent way for the user. The work is aided by modern and detailed APIs provided by RNTuple. In this paper we present the progress made and lessons learned.

1 Introduction

The typical data processing workflow of the ATLAS experiment [1] consists of multiple steps. After collision data has been recorded or Monte Carlo data has been generated, it undergoes further processing procedures such as reconstruction, derivation, analysis or simulation, digitization etc. ATLAS stores the output of all the data processing steps, except for collision data, in ROOT [2] files. At the end of the corresponding step, each produced ROOT file is validated by dedicated tools. Additional tools are used by the Software Performance Optimization Team (SPOT) as well as by users to summarize the contents of our data files. These tools were developed by the experiment as part of its software framework called Athena [3]. Since the beginning of LHC (Large Hadron Collider) Run 1, ATLAS has used the ROOT TTree format to store its data. Hence the tools were developed to work with this particular format. The ROOT project [4] has recently developed a new Input/Output (I/O) subsystem as replacement for TTree, called RNTuple [5]. As ATLAS will be adopting RNTuple for LHC Run 4, the tools have to be adapted to also handle the RNTuple format transparently while preserving their functionality. This article describes the work done in this direction.

2 ATLAS event data model

The ATLAS Event Data Model (EDM) provides a collection of C++ classes that define detector/physics objects to allow efficient processing of complex algorithms. For example, silicon

**e-mail: Grigori.Rybkine@cern.ch

hits in the inner detector and detector hits in the muon spectrometer are fitted to make Tracks, energy deposits in the calorimeter cells are grouped to make Clusters, and these can then be combined to construct higher-level objects such as Electrons, Photons, and Muons. These objects have their associated classes in the EDM.

Part of the EDM is separated into transient and persistent, the so-called Transient / Persistent (T/P) separation. This allows a more complex transient data model necessary for efficient data processing and a simplified persistent data model reducing storage space required. The separation also allows for the so-called schema evolution, i.e., multiple versions of persistent data, while data processing always uses the latest transient data.

Further, parts of the EDM were rewritten mainly to unify reconstruction and analysis data formats and the result is now referred to as the xAOD EDM. The xAOD EDM does not need T/P separation and adopts a versioning approach to schema evolution, e.g., `xAOD::Electron_v1`.

For each event, objects belonging to the same category or group are placed in containers, e.g., `xAOD::Electron` in an `Electrons` container and stored, e.g., in a file, by the ATLAS Input/Output system.

3 File validation

File validation is an integral part of our production data processing workflows. A typical production job is steered by the configuration written in the Python programming language. The job reads input file(s), e.g., in AOD (Analysis Object Data) format for the Derivation step, executes the event-loop by invoking the C++ code via the Python bindings, and produces output files, e.g., in DAOD (Derived AOD) format for the Derivation step. File validation tools are implemented as Python modules containing various functions. Each production job loads such modules and invokes relevant functions on each output ROOT file. Within these functions, we check the type of the data object being read in order to distinguish between TTree and RNTuple cases. This allows for data written in either format within the same file and is transparent to the user.

For basic validation, we verify whether the data is readable. One way of doing that is the following procedure:

1. read/load a whole entry/event
2. repeat for each entry/event

In the case of data in the RNTuple format, we open the ntuple with `RNTupleReader` and then loop over the entries calling `LoadEntry` on the reader. This fetches the entry from the storage, decompresses the entry and maps it into the memory making it ready for processing.

As a complementary approach, we read the data column-wise, that is by branch for TTree or by field for RNTuple. RNTuple provides the possibility to read all values of a field within a cluster and store them in an array, the so-called bulk reading:

1. for a top level field bulk read all values belonging to a cluster
2. repeat for each cluster
3. repeat the above for each top level field

This way of reading is similar to what is done in the becoming popular so-called columnar analysis.

4 Data content summary

Inspecting and summarizing data content is a recurrent task. In particular, SPOT runs daily production jobs to monitor the sizes and breakdown of each official data format as the software evolves and find degradation in certain domains, not limited to I/O, developers verify the impact of their changes on the data files, and users obtain an overview of the files they are analyzing.

One of the heavily used tools for such a task is `checkxAOD.py`. At the heart of the functionality it provides lies the `PoolFile.py` module which is a collection of classes and functions that allow the retrieval of necessary information from ROOT files, more specifically, from TTree and now also RNTuple objects. As in the case of file validation tools, we check the type of the object being read to distinguish between the two transparently to the user.

In the case of RNTuple, we have made use of RNTupleInspector, RNTupleDescriptor, and RFieldDescriptor API, provided with the ROOT framework, to obtain on disk and in memory size of objects:

Listing 1. Snippet of PoolFile.py

```
inspector = RNTupleInspector.Create(ntuple)
descriptor = inspector.GetDescriptor()
fieldZeroId = descriptor.GetFieldZeroId()
for fieldDescriptor in descriptor.GetFieldIterable(fieldZeroId):
  fieldId = fieldDescriptor.GetId()
  fieldTreeInspector = inspector.GetFieldTreeInspector(fieldId)
  diskSize = fieldTreeInspector.GetCompressedSize()
  memSize = fieldTreeInspector.GetUncompressedSize()
  typeName = fieldDescriptor.GetTypeName()
  fieldName = fieldDescriptor.GetFieldName()
```

`checkxAOD.py` summarizes the sizes by data object type and category. Below is an excerpt from such a categorized data content summary:

```
================================================================================
       Event data
================================================================================
   Mem Size       Disk Size        Size/Evt      Compression  Items  Container Name (↴
       ↳Type)
--------------------------------------------------------------------------------
  ...
891573.209 kb    97321.384 kb        0.973 kb        9.161   100000  ↴
     ↳HLTNav_Summary_DAODSlimmed (DataVector<xAOD::TrigComposite_v1>) [Trig]
452467.007 kb   105341.146 kb        1.053 kb        4.295   100000  ↴
     ↳METAssoc_AntiKt4EMTopo (xAOD::MissingETAssociationMap_v1) [MET]
579608.195 kb   131559.247 kb        1.316 kb        4.406   100000  ↴
     ↳TruthHFWithDecayParticles (DataVector<xAOD::TruthParticle_v1>) [Truth]
361787.939 kb   136287.961 kb        1.363 kb        2.655   100000  Photons (↴
     ↳DataVector<xAOD::Photon_v1>) [egamma]
436486.695 kb   139395.710 kb        1.394 kb        3.131   100000  Electrons (↴
     ↳DataVector<xAOD::Electron_v1>) [egamma]
329938.066 kb   178593.521 kb        1.786 kb        1.847   100000  GSFTrackParticles↴
     ↳ (DataVector<xAOD::TrackParticle_v1>) [egamma]
408230.738 kb   198797.508 kb        1.988 kb        2.054   100000  egammaClusters (↴
     ↳DataVector<xAOD::CaloCluster_v1>) [egamma]
484725.909 kb   224840.447 kb        2.248 kb        2.156   100000  ↴
     ↳InDetTrackParticles (DataVector<xAOD::TrackParticle_v1>) [InDet]
958744.790 kb   301980.340 kb        3.020 kb        6.486   100000  ↴
     ↳AntiKt4EMPFlowJets (DataVector<xAOD::Jet_v1>) [Jet]
691336.940 kb   327884.061 kb        3.279 kb        5.158   100000  AntiKt4EMTopoJets↴
     ↳ (DataVector<xAOD::Jet_v1>) [Jet]
--------------------------------------------------------------------------------
786979.289 kb  2909231.332 kb       29.092 kb        0.000   100000  Total

================================================================================
```

```
      Categorized data
================================================================================
   Disk Size         Fraction    Category Name
--------------------------------------------------------------------------------
     0.026 kb        0.001       PFO
     0.055 kb        0.002       MetaData
     0.492 kb        0.017       BTag
     0.511 kb        0.018       EvtId
     0.549 kb        0.019       *Unknown*
     0.824 kb        0.028       Muon
     1.635 kb        0.056       Trig
     1.775 kb        0.061       tau
     2.078 kb        0.071       MET
     2.652 kb        0.091       InDet
     3.277 kb        0.113       Truth
     6.571 kb        0.226       egamma
     8.646 kb        0.297       Jet
    29.092 kb        1.000       Total

================================================================================
  ...
```

The corresponding SPOT plots are shown in figures 1 and 2.

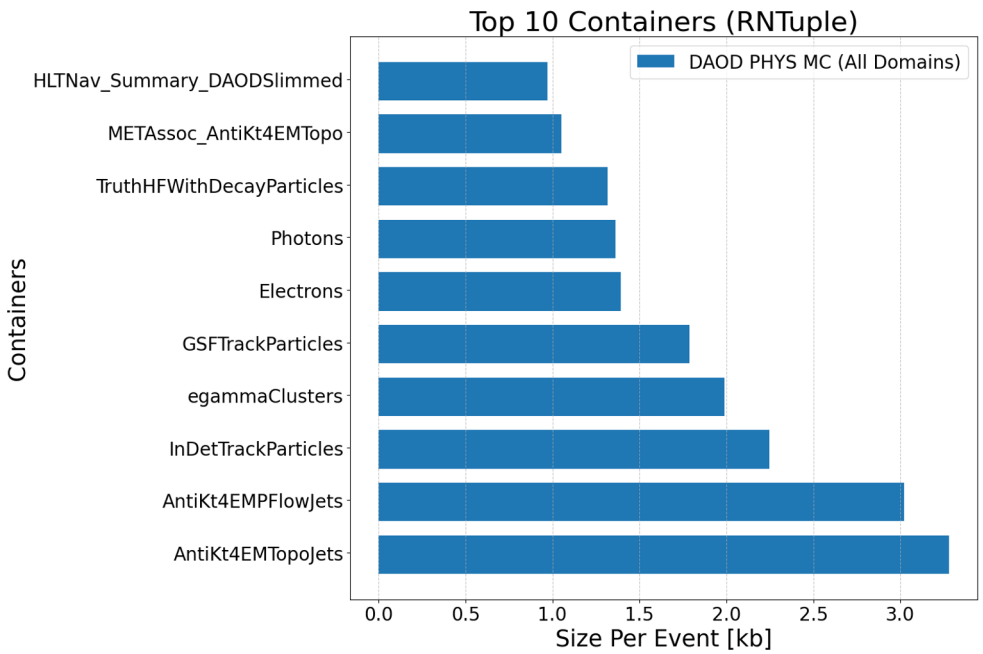

Figure 1. Size per event for top 10 containers in DAOD PHYS

It turned out that sometimes seemingly spurious numbers were reported for RNTuple, e.g.,

```
  9389707.954 kb   18014398508157516.000 kb 90071992540.788 kb        0.000    200000 ↴
      ↳HLTNav_Summary_DAODSlimmed (DataVector<xAOD::TrigComposite_v1>) [Trig]
```

After investigation, an issue and a pull request to fix it were opened. With the pull request accepted, the problematic line becomes

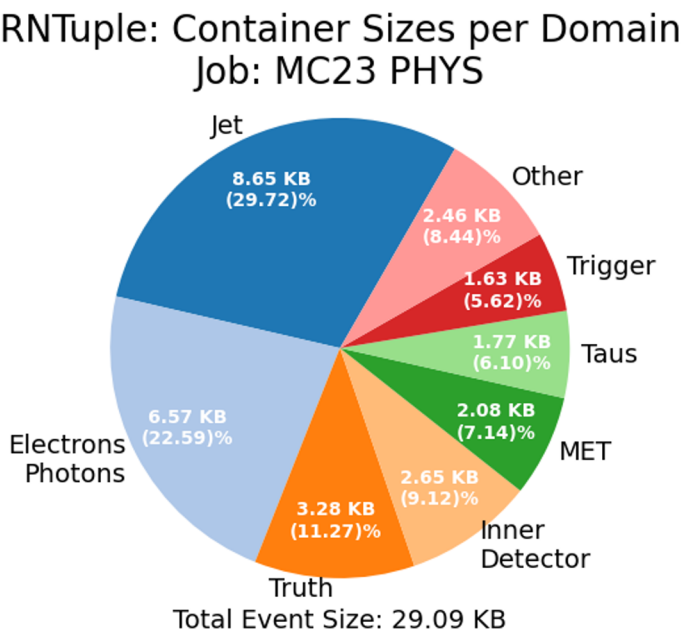

Figure 2. Container sizes per domain/category in DAOD PHYS

```
9389707.954 kb  2869832.889 kb       14.349 kb        3.272   200000  ↴
    ↳HLTNav_Summary_DAODSlimmed (DataVector<xAOD::TrigComposite_v1>) [Trig]
```

5 Conclusions

Several important ATLAS software tools for data file inspection have been adapted to allow for data recorded in ROOT RNTuple format. The resulting code is clear and robust thanks to convenient APIs such as RNTupleInspector. A contribution to ROOT was made to resolve the encountered issue. Work is ongoing to adapt the remaining tools.

The authors are grateful to Tatiana Ovsiannikova for providing the SPOT plots [6].

References

[1] The ATLAS Collaboration et al., "The ATLAS Experiment at the CERN Large Hadron Collider," *JINST* **3**, S08003 (2008). https://doi.org/10.1088/1748-0221/3/08/S08003

[2] R. Brun, F. Rademakers, "ROOT — An Object Oriented Data Analysis Framework," *Nucl. Inst. & Meth. in Phys. Res. A* **389**, 81-86 (1997). https://doi.org/10.1016/S0168-9002(97)00048-X.

[3] ATLAS Collaboration, "Athena," *Zenodo*, (2019). https://doi.org/10.5281/zenodo.2641996

[4] R. Brun, F. Rademakers et al., "root-project/root," *Zenodo*, (2017). https://doi.org/10.5281/zenodo.848818

[5] J. Blomer, P. Canal et al., "ROOT's RNTuple I/O Subsystem: The Path to Production," *EPJ Web of Conf.* **295**, 06020 (2024). https://doi.org/10.1051/epjconf/202429506020
[6] T. Ovsiannikova, A. S. Mete, M. Nowak, P. Van Gemmeren, "Impact of RNTuple on Storage Resources for ATLAS Production," (2024) *(in these proceedings).*