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

Marcin Nowak (BNL),

Peter Van Gemmeren (ANL), Alaettin Serhan Mete (ANL),
Tatiana Ovsiannikova (University of Washington)

On behalf of the ATLAS Computing Activity

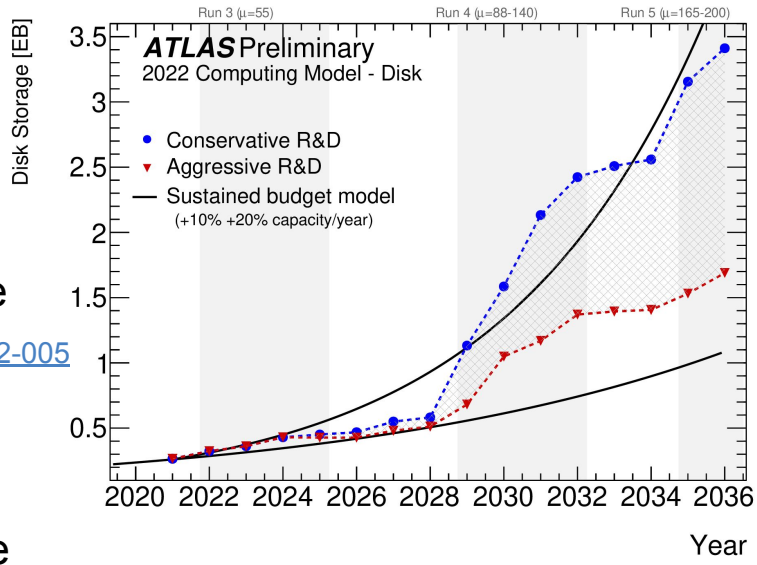
CHEP'24, 19-25 Oct, Kraków, Poland

@BrookhavenLab

General Introduction

- ATLAS - the biggest of the 4 particle detector experiments at LHC, CERN
  - Formed in 1992, taking data since 2009, software copyright 2002
  - Currently operating in LHC Run 3
  - Preparing software for massive data rates increase in Run 4 (Hi Lumi LHC)
- Athena - The ATLAS software framework for data processing (RAW→DAOD)
  - 4 mil lines of C++ plus 1.5 mil lines in Python
  - Depends on many externals, notably on ROOT for data persistency
  - Executed in parallel modes to achieve greater performance: multi-threaded or/and multi-process
- Processed data is stored in ROOT files using TTrees
  - Complex C++ objects, user defined, mostly collections
- This presentation describes the process of introducing a new storage type for Athena - ROOT RNTuple

Why Migrate to RNTuple?



- Handling Hi-Lumi LHC data rate will be a challenge

  ATLAS Software and Computing HL-LHC Roadmap - CERN-LHCC-2022-005
  - Flat budget for storage hardware
  - We should improve on the software side
    - Modern software
    - Software able to utilize modern hardware
- RNTuple comes with many performance relates features:
  - Asynchronous I/O, SSD-optimized access patterns, zero-copy, object stores interface
  - Contemporary code design and API
- TTree will become a legacy format while RNTuple will have priority in updates and improvements
- ATLAS began investigating RNTuple for Athena in 2021

Migration Checklist

- EDM compatibility
  - Can it handle all Athena data types?
- API compatibility
  - Can it plug into the framework interfaces similar to the old system?
- Operational compatibility
  - Can it run in AthenaMT and AthenaMP modes?
- Performance
  - Do we get better file sizes, memory usage and CPU usage?
- Non-Athena access (Python, Analysis)

EDM Compatibility

- Ability to handle **all** object types Athena stores in output files - ATLAS used simplified persistent types for Run 1, when ROOT dictionaries and TTree were less powerful, but since Run 2 we rely on some advanced TTree features
  - Support for STL containers - almost exclusively std::vector (up to 3 levels of nesting)
  - Read rules - allowing execution of arbitrary C++ code when an object of particular type is read
    - Code provided in the dictionary XML file
    - Used for state initialization (e.g. smart pointers), schema evolution
  - User-defined collection proxies - application-defined collection management providing ROOT direct access to elements (xAOD Containers and DataVectors)
    - Avoids redundant (sometimes impossible) element copying when reading
  - Dynamic model extensions - adding new variables “on the fly” (xAOD)
- Direct pointers and polymorphism are not used in Athena persistency
  - RNTuple does not support them

API Compatibility

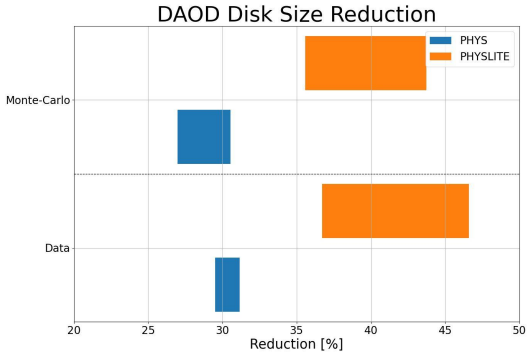
- Athena I/O layer was designed to make use of the original TTree API
  - Using void* and TClass for dictionary information
  - Not compatible with the modern, template based and user-friendly RNTuple API
  - Not even compatible with shared_ptr based API
- Athena relies on ROOT to create objects, when reading from a file, and to give up their ownership, so their lifetime can be controlled by the application
  - Not all objects can be copied
  - Creation “in place” is faster
  - Athena deletes Event data objects at the end of Event processing
- Read rules and collection proxies are not part of the TTree/RNTuple API - no chages
- Model extending has a new, RNTuple-specific API, but it can be encapsulated
- ATLAS intends to keep the current Athena I/O API for both TTree and RNTuple
  - Athena needs to be able to read both formats forever

Operational Compatibility

- Athena in production runs in AthenaMT and AthenaMP (DAOD) modes
- AthenaMP biggest performance challenge was efficiently merging output from all the workers into a single output file
  - Output merging is performed in-memory by the SharedWriter component, which can operate in 2 modes:
    - Legacy - with objects sent to the writer one by one and all TFile writing done by the writer process
    - Fast - with all workers independently writing to TMemFiles and sending fragments to the writer to be merged
- The Fast mode scales better, since compression is done in parallel on the workers
  - Typical job with 10 workers executes ~2x faster
  - This mode requires RNTuple merging capabilities that are still being developed (Legacy mode is operational)

Performance details presented by Tatiana Ovsiannikova in the talk:
“Impact of RNTuple on Storage Resources for ATLAS Production”

An example plot from this talk, illustrating file size reductions depending on the physics product, data type and data taking year



Performance

- File sizes
  - Observed size reduction for almost every type of data objects, except a single case of large nested vector<int>
  - Simpler data formats compress better: data PHYSLITE shows over 40% size reduction
  - Conclusion: bigger objects require larger compression buffers to better detect repeating patterns. RNTuple pages had up to now a rather small, fixed size
    - Very recently addressed by a new, adaptive algorithm to adjust page sizes
- Speed - no noticeable difference in typical production jobs
  - Not using the new advanced I/O features yet
- Memory usage
  - Somewhat higher memory usage than TTree (<10%)
  - Important goal - to fit into the 2GB per process limit for Grid jobs - work ongoing

Non-Athena Access (Python, Analysis)

See the poster by Gregori Ribkin: “ATLAS software tools to handle ROOT RNTuple”

- Python plays a big role in configuring of Athena jobs
  - Autoconfiguration reads in-file metadata from the input files
- After-job output validation is also python based
- TFile/TTree access from python uses TFile pythonization
- RNTuple access is slightly different (not TFile based)
- All Athena python scripts are now adapted to RNTuple
  - Still working on remaining standalone scripts
- Analysis access to Athena RNTuple files is also in development

Current Status

- Athena/RNTuple interface is implemented as a technology-specific plugin library
  - In fact a subtechnology for the ROOT plugin, also supporting TTree and TKey storage
  - Complete encapsulation
- Athena can choose storage format with a simple global job option
  - Choice even technically possible on a per stream/file basis
  - Transparent reading of both formats (cross-navigation possible)
- This flexibility and transparency is possible because the EDM and I/O compatibility allows using a single Athena build for both formats
- RNTuple interface is in the Athena production version
  - Based on LCG_106 software stack with ROOT 6.32, which now contains fully featured RNTuple implementation (before only found in ROOT HEAD)
- RNTuple tests are included in Athena CI test (automatically run in Git MR validation step) and also in bigger nightly ART tests

Summary

- ATLAS can write all official data products (RDO, HITS, AOD etc.) into RNTuple format
  - Took a long time and work on both sides and some fine tuning still needed
  - We are happy with the result
- Smooth transition between TTree and RNTuple formats, including coexistence
- We expect performance improvements
  - file size reduction - already observed
  - Execution speed, concurrency
  - Need to reduce memory usage
- RNTuple-related efforts moved from development phase to validation for production