Argonne NATIONAL LABORATORY

ATLAS

Peter van Gemmeren (ANL) for many

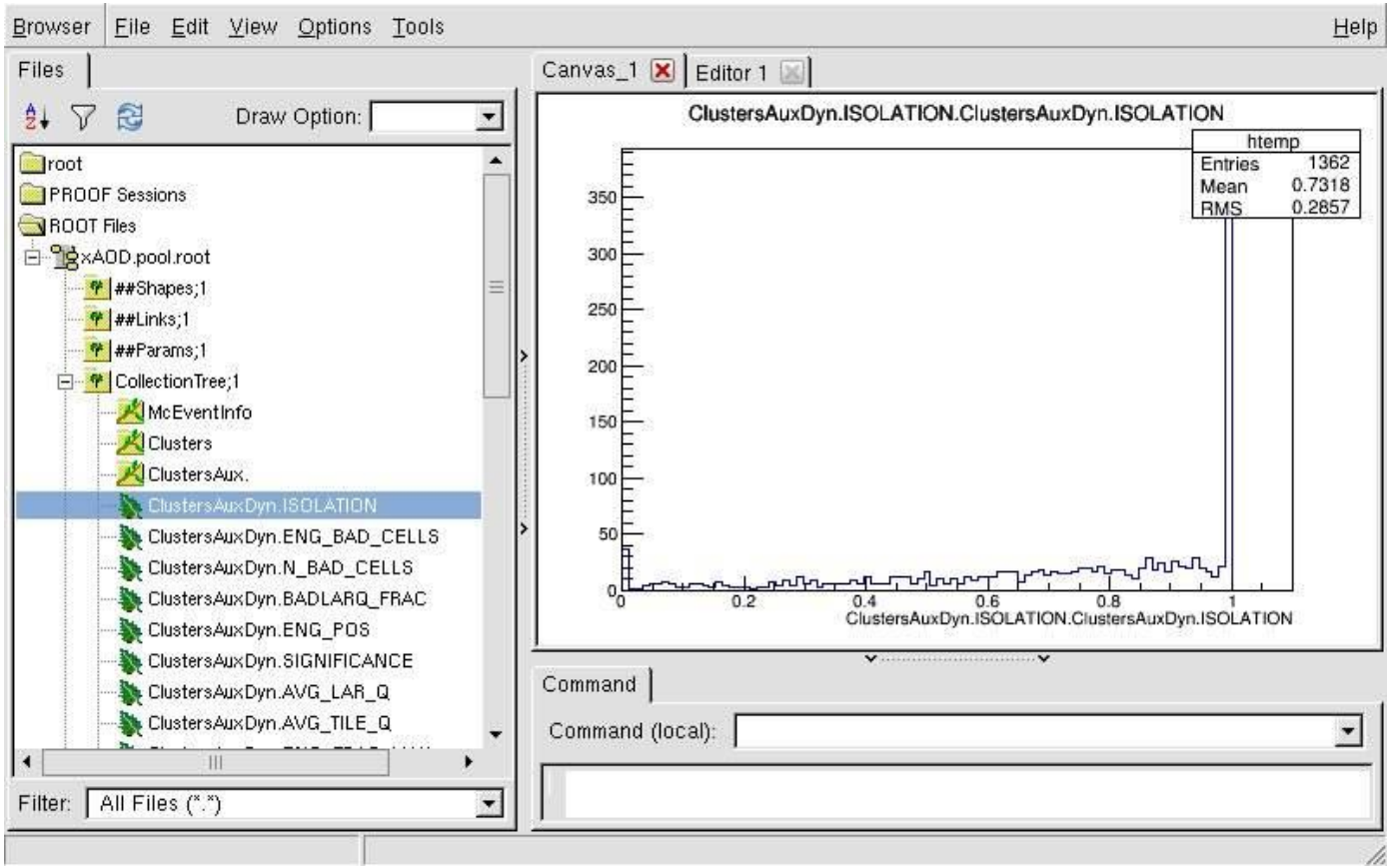
xAOD

- Replaces AOD and DPD
  - Single, EDM – **no T/P conversion when writing**, but versioning
    - Transient EDM is typedef of most recent persistent EDM
    - Versioning, similar to '_p<N>', but '_v<N>'
      - Limited support for schema evolution
  - Readable in Athena **and outside of Athena**
    - with a small amount of libraries loaded
      - xAOD files are browse-able in TBrowser without EDM libraries loaded

The xAOD design

- The new xAOD object has 3 components:
  1. xAOD **interface class**
     - Front end for the user, proper C++ object
       - May be without any data members
  2. Auxiliary Store - **static data container**
     - Predefined C++ type (similar to egDetails…)
       - Has a dictionary
  3. Auxiliary Store - **dynamic extension**
     - Dynamic structure, attributes can be added at any time
       - Extension to DataVector
       - No dictionary – needs special handling in the Persistency layer!
         - » The (former POOL) RootStorageSvc will assign a TBranch to each attribute.

ROOT browsing



Performance metrics

- xAOD EDM is done and will have some influence on performance, especially I/O.
- ROOT provides another layer of optimization that is crucial to I/O performance and has not yet been studied for xAOD:
  - Did so for AOD some years ago and saw good results
  - Write optimization parameters such as:
    - Streaming/Splitting, Basket Size/Auto_Flush, Compression…
  - Read optimization with:
    - TTreeCache
- Goal of performance tests will be to find good settings
- xAOD format is used at different stages of the workflow:
  - **Primary xAOD** (replacement of AOD) as **output of Reconstruction**.
  - **Derived xAOD** (~replacement of D3PD) as **output of TF2 derivation framework**.
    - Expect use-cases (and storage layout) to be different for Primary and Derived xAOD.
    - And don't forget upstream consequences of xAOD
      - E.g.: ESD/AOD shared data types!

Branches, Baskets and Compression for xAOD

- Primary xAOD in 19.0.2.1 has **2,119 Branches/Leaves**:
  - 204 core and other objects
  - 1,066 for concrete Auxiliary store objects
    - 50 objects, currently fully split*, but **[c|sh]ould be streamed member-wise** (at least for primary data)
  - 849 for dynamic Auxiliary store attributes
    - **Unfortunately these cannot be reduced.**
- Each Leaf has its own basket for compression and their size was optimized with **auto_flush = 10**, to hold a small number of events.
  - Good for primary data and event-wise reading.
  - Virtual memory needed for 10 events of decompressed data:
    - 4 MB for dynamic store, **33.5 MB for concrete store**.
- Compression factor for typical data is 3 – 4, anything much higher than that indicates redundant data, wastes CPU and memory.
  - 12 branches with **compression > 100**!
    - Needs fixing (e.g. each takes ~0.5 MB VMEM), **but number seems to have come down**.

Potential Concerns, Or Opportunities for Enhancement :-)

- Too many Branches could hurt read speed!
  - W/o cache or during learning phase each branch is equal to a disk read.
    - Caching kind of counteracts on demand retrieval of StoreGate
- Large Baskets consume lots of VMEM.
  - Surprised to find that even for small auto_flush of 10 events, total basket size was 40 MB!
- Highly compressible data will cause large Baskets, use up memory and CPU time.
  - Especially for downstream/analysis xAOD, where the auto_flush setting will be larger.
  - For combined ntuple writing they also seemed responsible for large jumps in memory consumption.
- Changes to xAOD have significant effect on ESD and need to be studied here as well.

Priorities for future ROOT Enhancements: Caching

- ATLAS data (especially xAOD) has to efficiently **support multiple use-cases**
  - Ranging for selective row-wise access (e,g., AthenaMP), to sequential row-wise access (e.g., production, TF2 derivation) to many column (e.g., Analysis) and few column-wise access (e.g., histogramming).
- **Caching** is an important ingredient to perform all these operations on the same data file and ATLAS has seen huge benefits from TTreeCache.
  - Work by DavidS on enabling cache by environment will ensure to maximize benefits.
- The following are ideas to enhance caching capabilities in ROOT
  - Read:
    - Hierarchical Cache:
      - long columns for few event/entry selection attributes in TTree, shorter columns for rest of accessed branches.
    - Smart (object aware) Caching?
  - Write:
    - Auto-Flush and cache for writing
    - IlijaV had provided better basket size algorithm, which should be evaluated / implemented.

Priorities for future ROOT Enhancements: Compression

- **Disk Storage** is already the biggest ATLAS computing expense and the main cost driver in any future upgrade. Decisions on reducing data content are unavoidable, but painful due to their impact on physics (and/or workflow).
- **Compression** is important for ATLAS to reduce redundancy in the data (current data has a compression factor of about 3 – 4).
  - ROOT Double32_t is being investigated to help restore some of the loss in storage efficiency by dropping T/P layer
    - Some double attributes in ATLAS EDM should be handled as Double32_t.
    - Need equivalent support for Float16_t as well
  - Support for varInt – like data type
  - ATLAS has not yet systematically studied LZMA versus current ROOT-provided compression

Error handling and related topics

- Trying to improve robustness and error handling wherever possible, support intelligent retry, and ensure that errors do not go undetected/unreported
- The latter is one component of a broader effort to improve monitoring and information flow among ATLAS layers
- ATLAS is relying increasingly on robust wide-area data access for analysis
- Have seen cases of undetected I/O errors
  - Andy H has pointed out examples of ROOT not checking for errors reported by plugin (xrootd) layers
  - And ATLAS code does not always check GetEntry() return codes
- There are circumstances in which it is possible that an error in one of N user-submitted jobs might go unnoticed unless she/he reads log files
- Possibly deep stack of components
  - Example: xrootd reporting to ROOT reporting to APR (former POOL layer) reporting to EventLoop or EventSelector or something else reporting to transform reporting to pilot reporting to …
- Aside: ROOT tutorial and other examples tend not to check GetEntry() return codes
  - What is standard practice here?

Outlook

- ATLAS switched to new xAOD data model, combining AOD and DPD
  - Simpler data model
  - More branches
  - No custom compression (T/P layer)
- Important future enhancements in the areas of:
  - Caching
  - Compression
  - Error handling / Robustness
- ATLAS is willing to provide effort to help with the development of some of the new features, where possible