xAOD persistence considerations: size, versioning, schema evolution

Joint US ATLAS Software and Computing / Physics Support Meeting

Peter van Gemmeren (ANL)

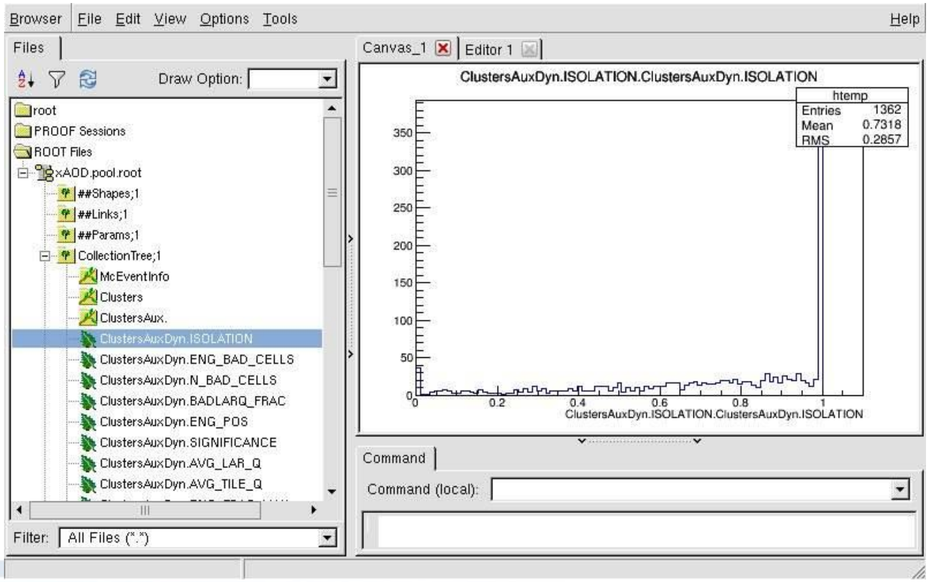
Outline

- Reporting on work done mainly by AMSG Task Force 1.

- What is xAOD
- Production and Derivation [TF 2].
- Event Data Model
- Persistent Layout
- Event Sizes
- Versioning
- Schema Evolution

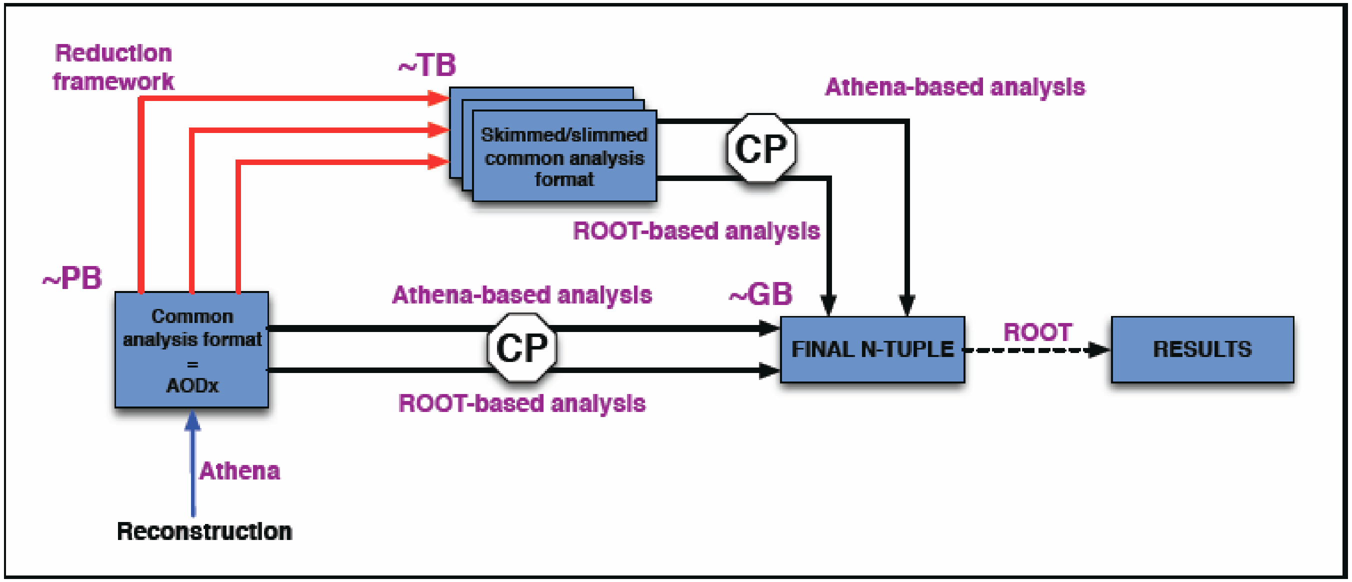
xAOD

- 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



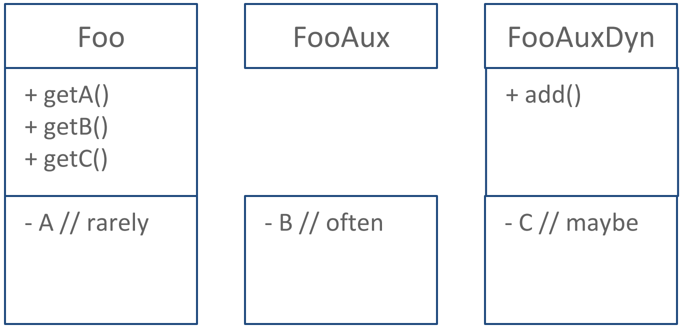
xAOD Production and Derivation

- [Athena] Reconstruction output will be [primary] xAOD.
  - Similar to current AOD
- No more copy/replication into [monster|primary] DPD.
  - This is the main source of storage savings
- New [Athena] Derivation framework to skim/slim data to [derived] xAOD.



xAOD Event Data Model

- 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.



Foo
+ getA()
+ getB()
+ getC()
- A // rarely

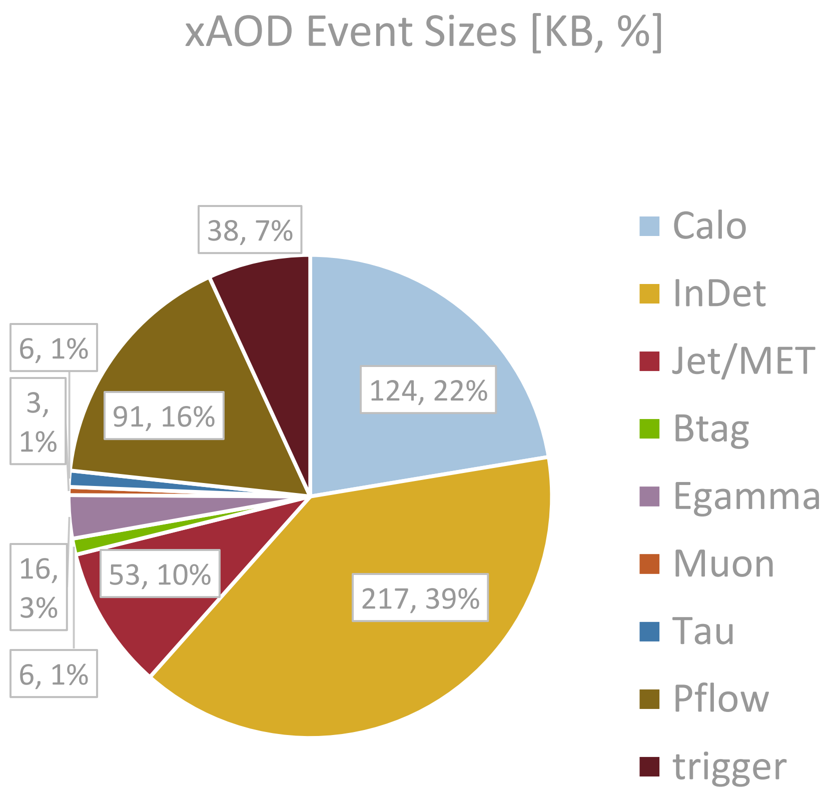
FooAux
- B // often

FooAuxDyn
+ add()
- C // maybe

xAOD persistent storage layout:
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
  - 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**!

xAOD Event Sizes



- Total Size 560 KB fur mu = 36
  - Up from ~420 for old AOD (same data)
- Most of the Data Model was completely redesigned and complexity/information/data content was reduced.
  - Except CaloTopo, which grew by a factor of almost 2, because of the lost of custom optimization in the T/P layer.
  - New objects, especially PFlow, contribute to the overall size increase.

Versioning

- Instead of complete transient persistent separation of all storable Event Data Model classes, a versioning approach was adopted.
  - T/P separation:
    - Each transient class foo has an independent (often much simpler) counterpart foo_p<N>, that stores the state (i.e. data member of foo).
      1. The persistent class can simplify (often overly complex) transient classes (no member functions, pointers, inheritance…).
      2. The persistent class can optimize the storage layout of a class (e.g. floats instead of doubles, vector instead of map…).
      3. A schema change that cannot be handled by ROOT automatically, can be made backward compatible be providing a new foo_p<N+1> and a read converter for foo_p<N> to the new transient foo.
    - In the past all three of these capabilities have been critical to ATLAS at some point.
  - Versioning:
    - Similar infrastructure, but foo is a typedef of the latest foo_v<N>.
      - No T->P conversion on writing, or when reading latest class version, but schema evolution can be accomplished using T/P converter.

Schema Evolution

- As in the previous Data Model, schema changes beyond the capability of ROOT can be handled using T/P converter:
  - A new class version _[v|p]<N+1> is introduced, along with a custom converter allowing to build it from a previous _[v|p]<N>.

However:

- The dynamic auxiliary store was intended to allow fast attribute selection and decoration.
- It does not have a predefined schema:
  - Runtime defined by modules calling addMember().
  - Can/Will change during event processing:
    - I.e.: If there are no Foo in the first events, there will not be any dynamic attributes corresponding to them (not even empty vectors, while there are empty collections), the attributes will be added on a later event, when the first Foo is found.
    - I/O infrastructure (such as reading, writing and merging), needs to be able to handle this.
      - E.g.: By backfilling data with ‘0’s.

No Schema Evolution on Dynamic Store

- As there is no defined schema for the dynamic store, there is no simple schema evolution (e.g. using T/P seperation).
  - Attributes are read on demand and exceptions are thrown if a requested attribute does not exist.
- After a mores complex schema change, [derived] xAOD can no longer be read and has to be reproduced.

Outlook

ATLAS Digest: Weekly news - 14 August:

- **The "AMSG Task Force #1" has concluded its work successfully.** Its mandate was to define and implement a new Event Data Model which can be used both within root and athena. This has been accomplished with the development of the xAOD which is now reality for run-2, and is undergoing large-scale user testing during DC14. Thanks to the conveners, Attila Kraznahorkay/CERN and Peter van Gemmeren/Argonne, and all the members (S. Binet/Orsay, P. Calafiura/LBNL, P.-A. Delsart, W. Lampl/U. of Arizona, R. Mandrysch/U. of Iowa, J. MItrevski/LMU Munich, D. Rousseau/Orsay, F. Salvatore/U. of Sussex, RD Schaffer/Orsay, S. Snyder/BNL)
- In a more private email from Beate Heinemann:
  - The activities continue in the Core SW and ASG groups, and your continued work in those areas is highly appreciated!

Upcoming work (incomplete list)…

- Event content optimization
- I/O performance tests and improvements
  - Memory consumption, read speed (CPU, wall clock), disk size (compression)…
  - Tuning of ROOT parameter, such as streaming mode, splitting, basket sizes and optimization, caching and others.
  - Some of these optimizations would require new core I/O functionality.
- Framework cleanup and completion
  - Some of the work done in TF1 was a little hasty and may benefit from a second inspection to clean up the architecture.
    - E.g.: Streaming of dynamic auxiliary store
  - Other developments are still incomplete
- Future xAOD work will blend in with ‘regular’ core and I/O activities
  - E.g.: ROOT 6 migration will have to deal with xAOD