Argonne NATIONAL LABORATORY

I/O Framework and Event Data Storage Developments

Peter van Gemmeren

U.S. DEPARTMENT OF ENERGY

Outline

- AthenaEventLoop
  - DataHeader Satellite
  - EventInfo Incident
- Data Layout
  - Run 1
  - ESD
  - Primary AOD

Athena EventLoop and DataHeader Satellites

- In Athena I/O, the DataHeader is the most fundamental object:
  - Stores StoreGate state and persistent addresses for all other objects and container.
    - With hundreds of objects this becomes rather sizeable
  - DataHeader also stores event provenance in support of back-navigation.
  - Supports creation of object proxies and their retrieve
- For full event reading, DataHeader performance impact is small <5%, but if only little (or no) data is read the DataHeader creates a major slow-down
  - Always needed, because Control Framework retrieves EventInfo, just to iterate over events.
    - To read any object, you need the DataHeader, which stores hundreds of elements for which hundreds of proxies are created/reset.
- Idea DataHeader Satellite: Create a DataHeader, that only stores elements for a small (frequently needed) subset of objects and container plus reference to full DataHeader.
  - Infrastructure, first implemented in early rel. 18 and fixed (for reading) in dev.
  - Currently writes “basic” satellite with just EventInfo reference.

- So DataHeader Satellite writing has been in place silently since start of Run2.
  - No defects or resource use that bother anyone
- But so far not used
- Until Will asked me about Athena bare-bone ‘not doing anything’ job being too slow and I remembered…

- First, Will got a point (here is what I see, but his numbers are similar)
- AthenaEventLoop, not doing anything, even MsgSvc turned way down goes at about a few hundred Hz:

```
cObj_EventInfo#...    INFO Time User    : Tot= 1.34  [s] Ave/Min/Max= 79.9(+-2.18e+03)/    0/2.24e+05 [us] #=16764
cObj_DataHeader...    INFO Time User    : Tot= 14.6  [s] Ave/Min/Max=0.871(+- 0.59)/    0/   10 [ms] #=16764
ChronoStatSvc         INFO Time User    : Tot= 50.4  [s]                                                #=  1
```

  - And that already may seem fast to some, but reading DataHeader took 30% of CPU time.
    - In fact the job is slower due to initialize etc

*Caution PerfMon 'evt‘ times don’t include Framework, just AthSeq (no DataHeader, EventInfo read or Proxy creation/reset). That’s why I use ChronoStat.

- Ok, so lets try the existing DataHeader Satellite:
  - Has only EventInfo reference, but that is all we need.
  - It is much smaller and therefore should be faster.
  - Simple jobOption config: `svcMgr.EventSelector.CollectionTree = "POOLContainer/basic"`

```
cObj_DataHeader...    INFO Time User    : Tot=  145 [ms] Ave/Min/Max= 8.65(+-  105)/    0/4e+03 [us] #=16764
cObj_EventInfo#...    INFO Time User    : Tot=0.681  [s] Ave/Min/Max= 40.6(+-2.24e+03)/    0/2.26e+05 [us] #=16764
ChronoStatSvc         INFO Time User    : Tot= 15.7  [s]                                                  #=  1
```

  - Nice Improvement:
    - DataHeader read speed goes way up, as it only holds ~1% of data
    - Overall time goes down even more, as Control framework has to handle only <1% of proxies
- Athena now running at ~kHz Event Iteration and DataHeader is no longer slowest.
- Can access full event content, but that causes slow reading of full DataHeader.
  - May still get savings, as proxies are now generated on demand.
- Now EventInfo is slow

*I got no idea why EventInfo reading seems to have sped up with the satellite, but it seems reproducible.

EventInfo retrieve for BeginEvent Incident

- That reminded me of still to-do work, avoiding control framework reads of EventInfo if information is available as in-file TAG content.
- Since Start of Run2 ATLAS is writing small TAGs directly into xAOD files:
  - Contain: EventWeight EventNumber RunNumber LumiBlockN EventTime BunchId
  - Should be enough for EventInfo needed by framework.
    - Still checking Incident clients…
  - Code currently tested in mig4:

```
cObj_DataHeader...    INFO Time User    : Tot=  122 [ms] Ave/Min/Max= 7.28(+- 88.4)/    0/3e+03 [us] #=16764
ChronoStatSvc         INFO Time User    : Tot= 14.1  [s]                                                #=  1
```

- Speed-up by a little, no more EventInfo reads at all…
  - One more trick from Will (Totally switch off AthenaEventLoopMgr Printout)

```
cObj_DataHeader...    INFO Time User    : Tot=0.786  [s] Ave/Min/Max= 46.9(+-  224)/    0/5e+03 [us] #=16764
ChronoStatSvc         INFO Time User    : Tot= 12.9  [s]                                                #=  1
```

  - Athena at well over 1KHz
- But the EventInfo incident is still under development.

*I got no idea why it seems to have slowed down the satellite, but it seems reproducible.

Data Layout: Run 1 recap

- For ESD and AOD, data was accessed via Athena (StoreGate).
  - Athena does event-wise processing.
  - StoreGate reads collections completely (all attributes), but on demand.
- Data Storage reflects that use-case
  - Each StoreGate collection is streamed member-wise into single (un-split) branch
  - This results in about 300 – 500 ‘sizeable’ branches.
  - ‘Sizeable’ means, that even with a small auto-flush setting of 5 / 10 events reasonable compression and I/O performance was achieved.
    - Memory requirements ~ 10 MB
    - Small number of total baskets -> small number of disk reads
      - And no problems keeping track of the baskets either…
    - Small number of events / basket -> efficient event selection
    - Member-wise streaming -> no efficient single attribute access

xESD

- As many data types are share with xAOD, the ESD data product changed substantially for Run 2
  - Now stored fully split (intended for xAOD browsing)
    - Between 5,000 and 10,000 branches
- But in **rel. 20.1** remains having a small auto-flush setting of 5!
  - Pretty obvious mismatch: Too many, too small baskets
    - Poor compression (1.x, excluding Pixel).
    - Significant over-head from ROOT to keep track of baskets

1. Reduce number of branches
   - Limited possibilities, changed static aux store, but dynamic aux store isn't streamed by ROOT and has variable content/schema.
2. Increase auto-flush setting
   - Not free, costs lots of memory (~un-compressed event size -> several MB per event)

- In **rel. 20.7** stream static aux stores member-wise and set auto-flush to 10
  - Big improvement for file size and read access speed.

Primary xAOD

PxAOD

- Now stored fully split (for ROOT browsing, efficient single attribute read)
  - Between 6,450 and ~10,000 branches
    - 274 core, 6176 static, <x> dynamic branches (<x> varies)
- Auto-Flush was raised to 100 (from 10)
  - Cost ~100 MB in VMEM, but large savings in storage disk (~20%) and faster read.
- Reasonable compression factor 3 – 4

- Still large number of small baskets not beneficial for storage size and disk access
- Analysis group now discourages direct ROOT reading of primary xAOD, so they don't need to be optimized for that use-case.
  - Member-wise streaming of core classes and static aux store
    - In 20.7,
      - saves 5 – 10% storage, with no change of data content
      - read speed seems better (10 – 20%), but needs more testing.
    - Splitting the largest container can save an additional 1 – 2%, but is waiting for 20.7 approval.

Outlook

- Features to speed up Athena EventLoop are in place and waiting for wider adoption and testing:
  - DataHeader Satellite (we could even write more)
  - EventInfo Incident (still under development, but TAG attributes are there for your enjoyment)
- KHz is not an ugly word in Athena.
- New data storage adjustments/repairs for ESD and primary xAOD are ready for December reprocessing
  - Should make significant savings in disk storage and better Athena access.

- Feedback is appreciated.

Backup

ROOT Splitting

- When assigning objects (or structs) to TBranches, ROOT can either stream all data into a single basket or assign separate baskets to each member (split the object).
- The Split-Level controls how deep in class hierarchy ROOT will assign individual baskets:
  - Split-Level = 0: No splitting, all content is stored in a single basket.
  - Split-Level = 99: Split into individual data members.
    - Typically better compression
    - Maximum number of baskets
      - Allows fine grained (column-wise) data access.
- ROOT also allows member-wise streaming into a single basket.
  - Different from just Split-Level = 0, as data is ordered by member rather than object.
    - Typically better compression (similar to Split-Level = 99)
    - Single basket (same as Split-Level = 0)

ROOT Auto-Flush

- ROOT persistence can compress and store data baskets in different ways:
  - Assigning a fixed basket size (e.g. 16K)
    - Data is written to disk for each basket, when that basket reaches its basket size
  - Auto-Flush after a fixed number of events/entries
    - ROOT optimizes the individual basket sizes depending on the data content (which may vary entry to entry)
    - Once it reaches the number of events, all baskets are written to disk
      - Build a ‘cluster’ which can enable efficient caching on read.
  - Using Auto-Flush with memory allotment
    - Similar to Auto-Flush with number of events, except number of entries is determined when memory allotment is reached the first time