METADATA STATUS AND FUTURE

Jack Cranshaw

Argonne National Laboratory

METADATA WORK AREAS

- Support for emerging workflows like Event Service and HPC.
- Metadata robustness and transparency.
- Reading and writing metadata/non-event data in an MT environment.

June Metadata TIM: https://indico.cern.ch/event/515805/

CURRENT FILE METADATA

- Standard Contents AOD/DAOD (<10 MB/file)
  - I/O metadata (EventStreamInfo, EventFormat)
  - IOVMetadata (TagInfo, Geometry, amitags, software versions, conditions versions and data, trigger)
  - Bookkeepers (luminosity, events) *rootcore*
  - ASG auxiliary metadata (trigger) (also exists in IOV) *rootcore*

- Processing in an athena job

Created on output:	i/o metadata, ASG AUX metadata.
Used by job, copied directly to output:	iovmetadata, asg aux metadata
Modified or created by job:	Bookkeepers, TagInfo

- Excel summary of file metadata contents
  - https://indico.cern.ch/event/515805/sessions/195447/attachments/1289964/1920667/ATLAS_Metadata_Size.xlsx

WORK FLOW PROPERTIES

	Workflow	Client	Job (Core/Reco/ASG)	Style (Prodsys)	Environment (Panda)	Resource (Panda)
Basic	**1. Standard prod**	Production	Serial	Single step	Grid	Guaranteed
	2. MP prod		MP			
	3. Athena user	User	Serial		Isolated/Grid	
	4. Root user		Root			
Advanced	**5. HPC prod**	Production	MP		Isolated	
	6. Flex prod				Grid/Isolated	Opportunistic
	7. Multistep prod			Multi step	Grid/Isolated	Guar./Oppor.

Other coordinates
- *Data products*
- *Data/Sim*

SW Meetings

ADC Meetings

BOOKKEEPING AND INCIDENTS

- Possible Incident trains: BeginInputFile (BF), EndInputFile (EF), MetaDataStop (MS), EndRun (ER)
  1. BF EF BF EF MS *ER*
  2. BF MS MS EF BF MS MS EF MS *ER*
  3. BF MS MS *ER*
  - The only trains we're currently equipped to handle
    - … BF MS *ER*, generates *incomplete* collections
    - ... BF EF MS *ER*, generates *complete* collections
      - Where … indicates N x (BF EF)
  - Issues
    - Infrastructure does not handle cases with (MS MS …) needed by Event Service.
      - Currently handled using EventSelectorTool which clears all CutBookkeepers on first event <u>after</u> MetaDataStop.
    - No way to recover complete from incomplete in later stage, e.g. merging.
      - Open question. Document around it? Special merge tool which fixes any misassignments?

METADATA DURING MERGING

- Metadata merging uses athena so that it has access to the proper tools (metadata merging is not just stacking). This can be done while merging events (*standard merge*) or by skipping all events (*hybrid merge*).
- Metadata access during merge has been more of an issue than the metadata merging itself. This was particularly true for I/O limited activities such as DAOD merging. Two leading causes have been identified.
  - **Slow access to metadata by job configurations while merging.**
    - Using athena the way AthFile does has multiple issues: caching, multiple reads of identical info, …
    - Athena is an event processing, not metadata processing, framework. Like using a bulldozer to cross the street.
    - A slimmed down method of getting metadata and reduction of the metadata content have alleviated this problem ... for now.
  - **Compressing and uncompressing the same *event data* multiple times.**
    - With process then merge, (decom, com, decom, com). Better to avoid merging.
    - For single file inputs a merge is also needed to resize baskets for downstream reads.
    - Suggestion is to increase number of inputs. Fewer jobs and files. More events per file. Requires
      - Rearrangement of trains by selectivity. (Done)
      - Remote access to input for derivation jobs as there is not enough local staging space. (In progress)
- **Problems with root settings.**
  - Primary AOD and DAOD have different root settings. DAOD and AOD iuse different tools for stream management.
    - DAOD root settings are fine on output, but are reset to AOD settings during merge.
    - Best answer is to have a dediicated DAOD merge. (manpower limited)

Currently not metadata merging itself

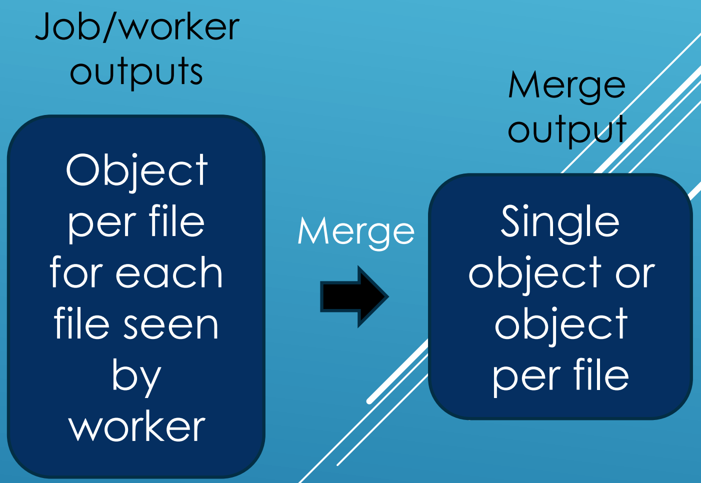
MERGING STRATEGIES

- Although merging of the metadata itself is not currently a performance bottleneck, it does complicate the process. Part of this is the chosen strategy.
- Merging strategies
  - *Data + Tools* (current): Objects are simple, just data. Policies for taking the union of two objects is contained in a tool.
    - Advantage: simple objects, policies can be updated separately.
    - Disadvantage: Must correlate tools/data and track tool versions.
  - *Merging methods/functions*: Objects know how to merge themselves.
    - Advantages: Merging can be made more automated.
    - Disadvantages: Changing policies may mean rewriting the data. Means rewriting existing classes.
- Perhaps the choice should be re-examined based on the current needs.
  - Some data products don't have long lifetimes, e.g. DAOD.
  - Policy evolution is normally to fix bugs, and this may/should stabilize.

DEALING WITH GLOBAL STATE, INCOMPLETE/COMPLETE

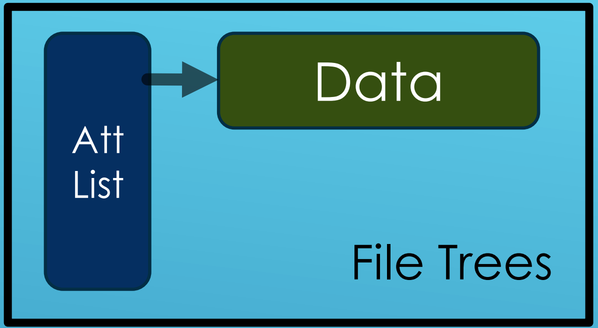
Complete/incomplete decisions couples subjobs to a global state.

- *Dumb down the subjobs (delay labeling till merging)*
  - Just let the subjobs count, but associate the count with a key and put them in a container
    - Cont<file, counts>, cont<lumi, counts>, …
  - Define complete/incomplete with a query later
    - Compare <file, counts> in dataset to full file counts from rucio/AMI.
  - Advantages:
    - tools in subjobs have simpler state.
    - Global state can be defined at any point: file, dataset, analysis.
  - Disadvantages:
    - Global state not always known by just looking at file.


Job/worker outputs
Object per file for each file seen by worker
Merge
Merge output
Single object or object per file

EVENT LISTS VS COUNTS: EVENT STUBS

- At filter stages, don't throw away events, prune the data tree.
- We write an Attribute List with EventInfo (lite) in it that points to the event DataHeader (currently only used for AOD).
  - We could write POOLCollectionTree (Attribute List) entries for all input events and data (DataHeader/CollectionTree) entries only for accepted events.
  - Attribute list tree contains EventInfo information for both input and output events.
  - Attribute list tree could also contain simple skim decision information.
- Advantages:
  - ease of merging, just concatenate.
  - Can mark why events were rejected, catch duplicates.
- Disadvantages:
  - if you want counts from previous stages in the final data file, you probably need to create bookkeeper objects for those, so multistage bookkeeping is not covered.
    - Might not be needed if we exported sums to external database. (Event Index?) Bookkeeping has provenance, but is storing that in the file the correct strategy?



General design goal: when we build counts vs lists should be configurable.

TRANSPARENCY OF METADATA RETRIEVAL

- In-file metadata was added to allow off-grid processing, to act as a cache. This is how it is still used for conditions, but gradually a set of objects and users built up, which depended on in-file metadata directly without the indirection layer.
  - In-file metadata becomes critical component rather than optimization. Complicates merging due to its union vs. append nature.
- Can we regain the abstraction?
- Maybe we can do something like the IOV interface of conditions (file, lumiblock, …). A lot of work has been done in that area on process isolation and interfaces which could possibly be generalized.
  - Is the model where metadata is kept in-file only a good model, or can current in-file metadata be stored (and retrieved) in another technology?
- Benefits
  - Allows us to have different environments/optimizations for connected (production) and isolated (analysis) environments
  - Having data available elsewhere may help with file peeking performance issues.
  - Being able to transparently access a variety of metadata sources makes us more flexible.

MULTITHREADED METADATA

- The metadata infrastructure in athena(non-MT) has not been migrated to a multi-threaded environment. There are several challenges and developments pending in this area.
- Services must be made thread-safe.
  - MetaDataSvc, CutFlowSvc, DecisionSvc
- Metadata processing is driven by incidents. Incidents are inherently thread-unfriendly. Sami has been working on a model for handling incidents in AthenaMT.
  - Migrate (and/or give feedback) to the new incident model.
  - Review existing code and interfaces to minimize or eliminate use of incidents.
- For conditions the AthenaMT model has transformed many tools into algorithms for data dependency safety.
  - Should metadata tools be treated in the same way?
  - How does this work with dual-use tools which are what make our analysis model so transparent for CP corrections?
- Is there a model/architecture in which we can use the same interface for all non-event data (conditions, metadata, monitoring, …)?
- Unlike conditions, metadata will be collected and written in AthenaMT.

CONCLUSION

- Some near term milestones
  - Bookkeepers (event and lumi) completely working in Event Service (and other?) workflows.
  - Metadata optimized merging (accessibility, tools used, prescriptions by use case)
  - Thread-safe metadata services in current format (MetaDataSvc, DecisionSvc, CutFlowSvc).
- Some longer term milestones
  - IOV metadata more easily accessible outside athena (may be hard to make backward compatible)
  - Factorize global state out of single jobs and assign it during merging. (probably required at some level for event service).
  - Minimize use of incidents and add features to make metadata services thread friendly.
  - Develop architecture for metadata collection and writing in a multi-threaded environment.
  - Determine how much of the AthenaMT infrastructure for conditions can be reused for metadata and whether a general interface would work for all non-event data.
  - Discussions of interactions of in-file and external/non-filescale metadata and how they work together (Panda, object servers, AMI, event index, …).