Argonne NATIONAL LABORATORY

A Programmatic View of Metadata, Metadata Services, and Metadata Flow in ATLAS

David Malon*, Solveig Albrand, Elizabeth Gallas, and Graeme Stewart
for the ATLAS Collaboration

Computing in High Energy and Nuclear Physics 2012
New York, 21 May 2012

* malon@anl.gov

U.S. DEPARTMENT OF ENERGY

Not an outline

- What this talk is not:
  - Not an inventory of metadata
  - Not even a taxonomy
  - Certainly not about all information that could possibly be viewed as metadata
  - Not a description of specific metadata infrastructure components
  - For the above, see presentations and papers at this and earlier CHEPs
- Instead, a view of physics-related metadata and metadata flow as seen from the point of view of what happens just beneath the surface, even in something as rudimentary as execution of a single task reading one dataset as input and writing another as output
- Along the way, something about the tension between logical models and physical deployment, and how the difference is addressed
- And also along the way, something about the principles that have guided ATLAS decisions about metadata handling

Metadata access even before a simple task is run …

- Input dataset selection may have been based upon physics metadata
  - principal ATLAS repository is AMI
- Decisions about what to process and which auxiliary data to use to process it may have been made by consulting COMA
  - e.g., for trigger configurations and conditions data tags
- Data quality decisions about which subsets of which runs are appropriate to use for physics purposes will have been made and reflected in so-called good run lists

COMA Period Runs Report

Project Name (fnt) : data12_8TeV
Period Name (pn) : B5

Data Periods (1):

Found 2 Runs matching the input criteria.

Project Run	Run Links	StartTime	Duration	NLBN	SMK	Events	Period
data12_8TeV 203605	COMA; RQ; LumiDS	2012-May-19 03:19:19	20649 sec (5:44:09)	349 [1-349]	1396	2352882	AllYear, B, B5
data12_8TeV 203602	COMA; RQ; LumiDS	2012-May-18 11:39:11	54272 sec (15:04:32)	944 [1-944]	"	38345992	AllYear, B, B5

Pre-execution metadata access is not just about input datasets and auxiliary data

- Which software versions? What is an appropriate job configuration?
- For production, encoded in a configuration tag that is recorded both in AMI and in the output event data products themselves
- Example of a configuration tag: r2713_p705, which encodes information about
  - which ATLAS releases (17.0.3.3)
  - which database releases (16.9.1.1)
  - which transforms (reco_trf.py),
  - which job configurations
  - …
  - Employed in the (two) processing steps used to produce the data from the original raw data input.
- Details corresponding to these configuration tags are recorded in AMI
  - A service is provided to decode them
- Importantly, ATLAS physicists can configure their own subsequent jobs equivalently simply by providing the relevant configuration tag as an input argument

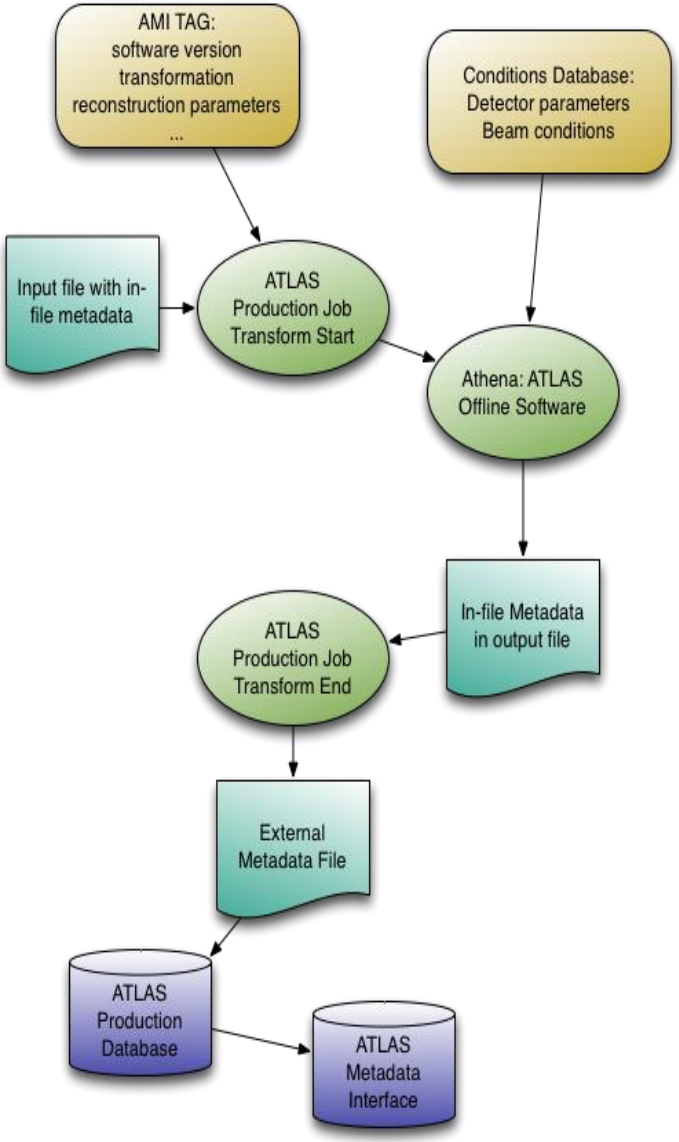
Pre-execution metadata access is not just about input datasets and auxiliary data

- Details corresponding to these configuration tags are recorded in AMI
  - A service is provided to decode them
- Importantly, ATLAS physicists can configure their own subsequent jobs equivalently simply by providing the relevant configuration tag as an input argument

tag	p705 Datasets
SWReleaseCache	AtlasProduction_17.0.3.3 SQFlags
DBRelease	16.9.1.1
Geometry	none GeometryInfo
JobConfig	none
transformation	Reco_trf.py
TriggerConfig	none TWIKI
ConditionsTag	none
description	
writeStatus	valid
readStatus	valid
createdBy	root
created	2011-09-12 09:11:59

Next? Job initialization is metadata intensive

- Algorithms and auxiliary services require access to non-event data in order to initialize themselves to process event data appropriately
- Some of this information comes from the configuration provided by the job transform itself or specified by the accompanying configuration tag
- BUT … some will depend specifically upon the input dataset.
- Such information may in principle be dataset-level metadata that could, for example, be extracted exactly once from AMI at task definition time and propagated to all jobs.
- Typical current practice in ATLAS, though, is to "peek" into metadata contained in input files
  - Possibly multiple times(!)


AMI TAG:
software version
transformation
reconstruction parameters
...
Conditions Database:
Detector parameters
Beam conditions
Input file with in-file metadata
ATLAS Production Job Transform Start
Athena: ATLAS Offline Software
In-file Metadata in output file
ATLAS Production Job Transform End
External Metadata File
ATLAS Production Database
ATLAS Metadata Interface

Peeking into input files … more than once?

- First for the purpose of job options configuration (before a reconstruction step begins, for example)
- Second for service and algorithm initialization within the reconstruction proper
- This practice provides a safeguard against certain kinds of configuration errors
  - of the sort that arise when one simply cuts and pastes job options used for processing a different dataset
- The capability to configure jobs automatically in this data-aware way is quite useful, but …
- it is also true that there are performance and consistency advantages to handling such configuration exactly once for all data within a dataset and not file by file or job by job

Metadata access during job execution

- Metadata such as detector conditions and calibrations that may vary over the course of a run may be accessed dynamically during event processing
  - either directly from databases or from caches derived and refreshed from such databases.
- There are input metadata transitions as well, arising, for example, when a job processes a sequence of input files.
- In this case the event loop and physics code do not care that a new file was opened, but certain auxiliary metadata may need to be updated or refreshed or flushed or accumulated or merged

Metadata propagation from input to output

- Certain metadata may be propagated from input to output, to be stored either in output data files or in metadata repositories such as AMI.
- Metadata may be created as well, and must be propagated in similar ways.
- Event counts, particularly when filtering is done, are a fundamental but important special case—more on this later
- In any case, machinery must be provided to emit metadata and to transport them from the producing job through the production system infrastructure to appropriate repositories

Logical versus physical constructs

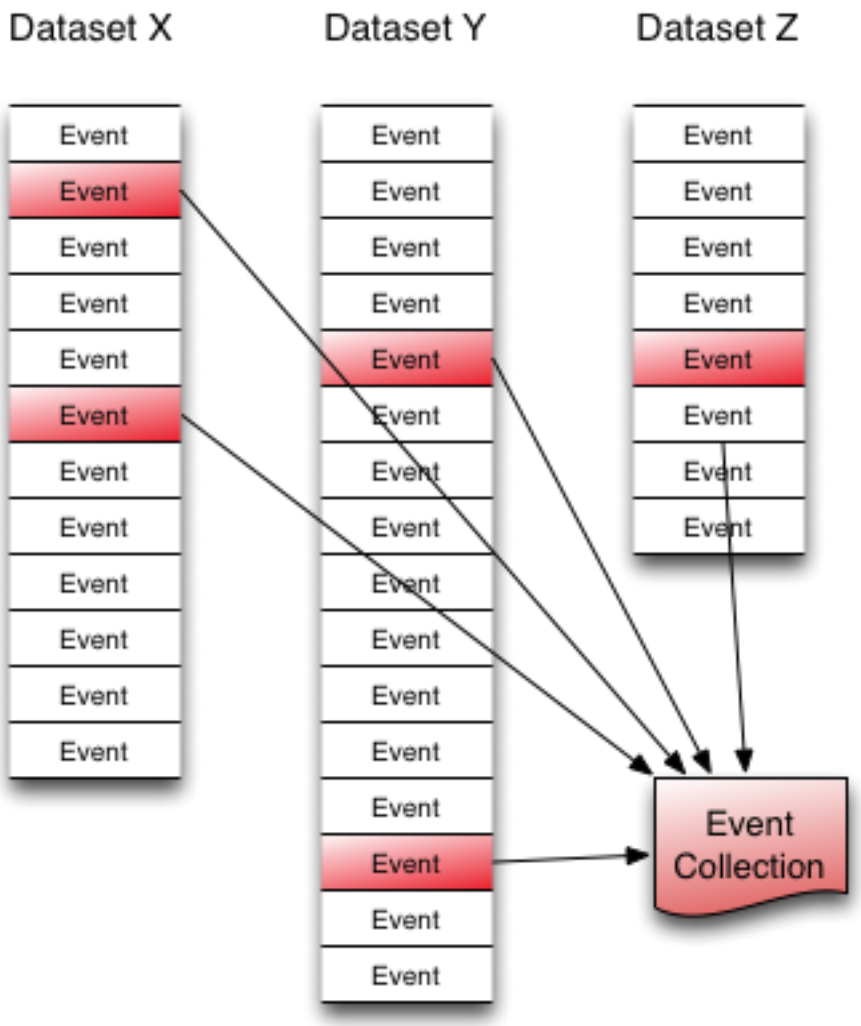
- Examples of logical, semantic units of data organization
  - Data-taking runs
  - Contiguous temporal segments within a run, known as *luminosity blocks*
  - The set of events that pass a particular trigger or suite of triggers
- Files, on the other hand, are artifacts of storage organization.
  - In most cases, a physicist needs to process a set of events corresponding to physics or data-taking criteria—for example, all the events from a given run that pass a given set of triggers—and whether those events are in one file or a thousand files, and whether those files also contain data from other events is, conceptually, a deployment detail
  - No resulting physics (or physics metadata) should depend upon storage organization
- A natural implication might be that there should be no physics metadata associated with files(!?), and in a sense this is true:
- In most cases, physics metadata about the file are really metadata about the collection of events *within* the file.
- This distinction might seem pedantic, but it turns out to be significant in a framework in which the I/O model is to process collections of events whether they are stored contiguously within a file or scattered across many files

Event collections and their metadata

- ATLAS I/O infrastructure supports the notion of event collections
  - much more directly than does the collaboration's distributed data management infrastructure
- ATLAS event selectors iterate over event collections that may be *explicit* or *implicit*,
  - *with explicit* meaning "this event list, or the set of events that satisfy this selection predicate"
  - and *implicit* meaning "the set of events that happen to reside in this input file"—the latter only implicitly (at least from the point of view of data semantics) defining a collection

Are datasets logical or physical constructs? Does it matter?

- Datasets provide something closer to a conceptual organization, but …
- even here the physical mapping implemented by ATLAS and other experiments corresponds to exactly one logical view:
- In practice, a dataset is, either explicitly or effectively, a collection of files.
- Example: the file set containing Analysis Object Data (AOD) for a single run and trigger stream and a given processing is a dataset in this data management sense
- One could imagine, though, that the subset of events *within* this dataset that pass a specific set of triggers is conceptually every bit as logical a candidate for "dataset" status
- And the fact that these events are mixed with events that passed other triggers that were written to the same stream is an artifact of storage


Dataset X
Dataset Y
Dataset Z
Event
Event Collection

When logical datasets do not map directly to storage datasets …

- While processing such samples is of course routine, they are not currently supported as datasets *per se* by ATLAS distributed data management catalogs
- Event collections instantiated as the output of TAG database queries are a surrogate for datasets of this sort and provide this functionality
- There have been discussions about "virtual datasets"—datasets that are well defined but not physically instantiated as separate entities in storage—and how one might support a more extended dataset functionality
  - not only in the I/O framework, as today, but in production data management as well
  - not currently a common practice in ATLAS to do this
  - though such desiderata are currently being considered as input to the next generation of distributed data management software

In-file metadata infrastructure

- ATLAS employs a sophisticated in-file metadata infrastructure that goes far beyond machinery to allow storage of non-event data in event data files
- Appropriate for mention here are the incident-handling features, as input file boundaries are artificial from the point of view of physics metadata, encountered asynchronously to state transitions of the event-processing framework
  - again, these artifacts of storage organization do not infiltrate ATLAS event and physics metadata handling
  - The architecture distinguishes the beginning and end of an input *collection* from the beginning and end of an input *file* even when those happen to be completely concurrent,
  - because it supports the more general case of an event collection incarnated as a list of references to events that may reside amid other events in multiple files
  - In such cases, the appropriate metadata to propagate to the output may not correspond to the complete metadata record of each input file, as only selected events are processed

Returned metadata

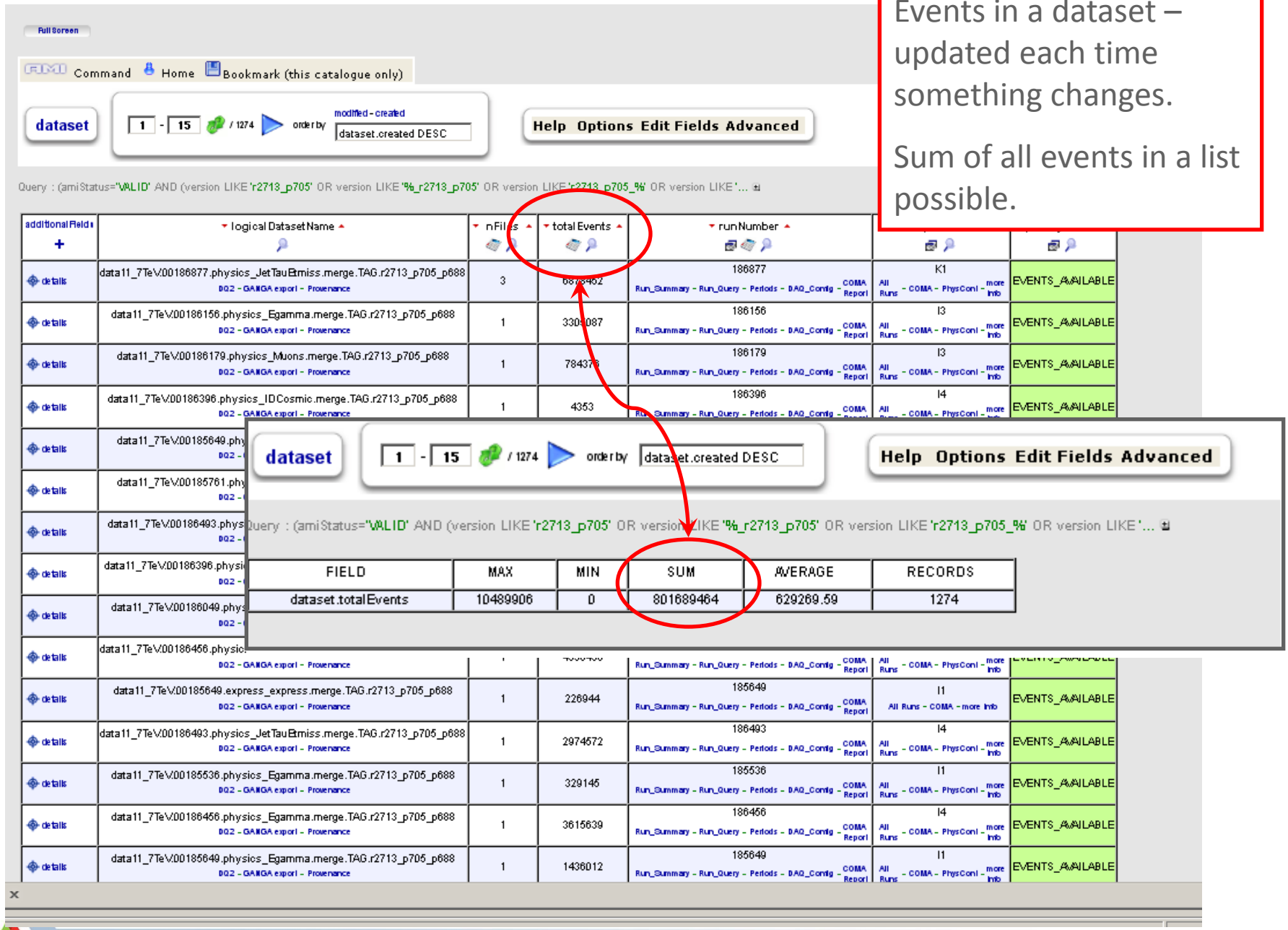
- Production transforms not only aggregate and propagate metadata to output data products, but they also return metadata for insertion into repositories for a variety of purposes.
- Elementary example: the list of files produced by a constituent job, their unique identifiers, and the datasets with which each of them is associated.
- That much would be required for correct task completion, but the files may also have semantic metadata associated with the event collections they contain, and such information should be returned as well.
- Special-purpose output metadata file with a format defined by an XML DTD
  - but has also used pickle files for certain information when metadata are returned in Tier 0 processing.
- Generally, physics metadata returned by jobs are routed unparsed into a production system database
- from which they are retrieved and processed for insertion into AMI
- In addition, an executing job may flag lines written into log files as containing information—{key, value} pairs, for example—to be included in the returned metadata.
- While this capability has been useful, metadata services are under development to provide a more explicit interface for this purpose

Event counting is fundamental

- Often provides a straightforward integrity check: does the number of events in the output match the number of events in the input?
- The question must be asked and answered, not only on a per-job level, but also on a task level, comparing input to output dataset event counts as a check that all relevant events in all relevant files have been processed.
- If a job filters its input ("skimming" events), did it at least read all of the intended input, and did it correctly record an output event count for downstream integrity checks?
- Production transforms routinely count events in output data products written in a wide range of data formats
  - and return event counts as semantic metadata associated with event output files

Events in a dataset – updated each time something changes.

Sum of all events in a list possible.



Should data management systems care about or maintain event counts?

- An event count is semantic information
  - as indeed is the fact that a file contains event data in the first place,
- Ergo one might expect that an appropriately generic distributed data management system should not care about such metadata
  - and that such values should appear “only” in AMI, which is the ATLAS repository for physics metadata about datasets and their constituents
- Event count, though, is important for task execution planning, because it is used to ascertain how many files should be input to each constituent job, or conversely how many jobs should be launched to process disjoint chunks of any single file
  - … so there is are pragmatic considerations to whether one should redundantly store event counts, say, in distributed data management catalogs
  - More on principles that guide ATLAS decision-making on such questions follows

Counting is subtler than that: some background

- A subtler and more complicated counting requirement comes from the need, in most cases, to process complete luminosity blocks.
- Many quantities relevant to event processing vary over time within a run
  - Conditions, calibrations, and machine luminosity are obvious examples
  - All are treated as essentially constant within a luminosity block
- Data quality assessments, too, are made at the granularity of a luminosity block
- Certain changes to the trigger configuration (e.g., changes to prescale factors as luminosity decreases) may also occur on luminosity block boundaries,
- … and event counting within the ATLAS multi-level trigger is also managed at the granularity of luminosity blocks.

Luminosity-block-level counting

- For all of these reasons, it is essential that an analysis know whether it has seen all qualifying events within a luminosity block,
  - for example to get the denominator right in a cross-section or other rate calculation.
- It is possible to add or drop complete luminosity blocks from an analysis and to handle the resulting statistics correctly,
- but it is much more difficult when one has seen only some of the events in a luminosity block,
- and the problem is particularly insidious if one does not know that this has happened.
- ATLAS attempts wherever possible to store events from an integral number of luminosity blocks contiguously whenever possible
  - Not at all unique to ATLAS, of course, but there are subtleties

Counting is subtler than that (event collections, remember, not just files)

- The metadata infrastructure propagates to downstream data products the list of complete luminosity blocks that have been seen by any given job
- This is straightforward to accomplish when luminosity blocks are integrally contained in input, as long as jobs read every event in their input
- The problem is slightly subtler when the event collection is the list of events returned by a query to the ATLAS TAG database.
- In this case, the list of luminosity blocks that constituted the domain of the query must be propagated, rather than the list built from pointed-to input files, and such propagation must be corrected if any files containing pointed-to events are unreachable or unreadable

Incomplete luminosity blocks

- .When luminosity blocks are incomplete in any single file—as may happen with raw data from high-trigger-rate streams, for example, because the trigger writes multiple files in parallel and they may be too large for practical reasons to merge into a single file—much more careful accounting (and event counting) is required to ensure that complete luminosity blocks have been processed.
- While this is relatively straightforward in the controlled context of Tier 0 processing, where access to information regarding how many events the trigger system believes it has written is available, this requirement places demands downstream as well
- Example? Merging of derived data files from multicore processing,
  - another scenario in which multiple writers may have seen only a subset of a luminosity block even though that luminosity block may have been complete on input
- Indeed, because merging metadata requires semantic knowledge not required in simple concatenation of arrays of events, correct and robust metadata handling is often the principal source of complexity in efficient merging of event data files

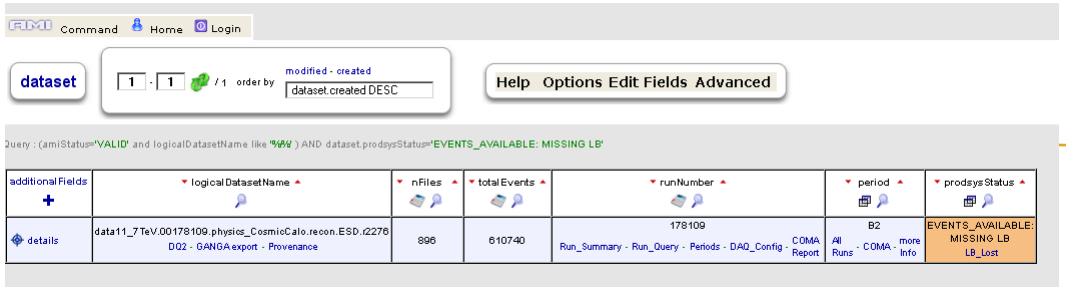
A principle and its converse

- If an event data file is unreachable or unreadable, it must be possible to ascertain what one is missing
- Conversely, for routine use it should not be necessary to consult an external metadata source to understand a file's content
- The first statement ensures the integrity of an analysis when it must proceed without some relevant data—it must be possible to determine precisely what one is missing, both to assess the significance of the omission and to correctly adjust statistics and cross-section calculations accordingly.
- The second ensures that all analysis needn't cease when one is on an airplane, or when a network connection to a remote database server is down.

Implications

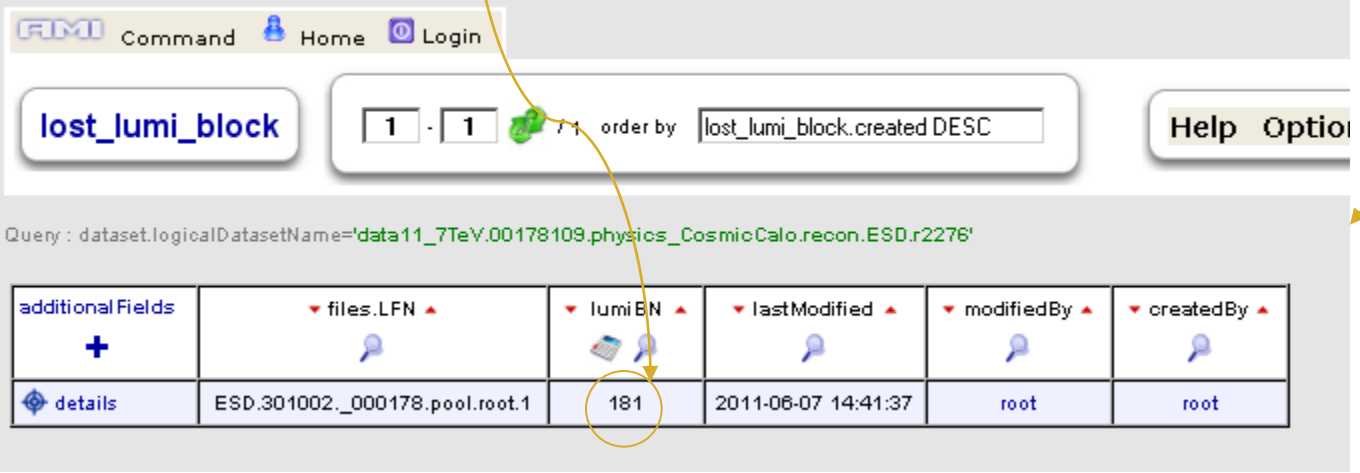
- While these principles sound straightforward enough, the first has stringent implications for bookkeeping, and for luminosity-block-level accounting and management
- A related consequence is that ATLAS takes pains to avoid splitting luminosity blocks across file boundaries whenever possible, and scrupulously accounts for the cases in which this cannot be accomplished
- The converse half of the principle guides decisions about what subset of metadata are stored in event data files,
- … and following this principle is what allows file peeking utilities to support correct configuration of jobs that read ATLAS data

Lost LB tracing



files

fileStatus	LOST
LFN	ESD.301002._000178.pool.root.1
events	741
fileGUID	EAC6DCA7-6A64-E011-9E0F-003048649D30
inputfile	data11_7TeV.00178109.physics_CosmicCalo.merge.RAW._lb0181._SFO-ALL._0001.1
lumiBN	
jobexeid	147409970
lastModified	2011-05-12 10:02:38



Design considerations: mutability, pervasiveness, frequency

- Applied in decisions about in-file metadata content, but much more broadly applicable
- Mutability: if metadata are likely to change, then they may not be good candidates for caching within event data files because of the risk that physicists may use outmoded values unwittingly
- Pervasiveness: refers to how widespread the need for the metadata might be
  - if they are needed only rarely or by a few analyses, caching may not be called for or worth the additional storage cost
- Frequency: refers to the number of times the information will be accessed
- If every analysis needs the information, but only once at the point at which their samples are being defined, then the requirement is pervasive but not frequent
- If the information is needed every time the file is opened but only for exceptional specialist purposes, then the requirement is frequent but not pervasive

Redundancy

- Storing equivalent metadata in multiple places, while sometimes convenient, poses certain risks.
- In some cases, the question of whether a metadata instance is redundant is ambiguous—a "view" (in the database sense) of metadata extracted from another source, for example, is arguably redundant,
- … but if the view is time-consuming or expensive to repeatedly reproduce or access, then it may be worthwhile to maintain the derived view as separate metadata.
- ATLAS decisions regarding metadata redundancy are guided principally by these pragmatic concerns
- … but a design principle is that even in these cases, it must always be clear what the authoritative source or repository of information is

Conclusions and future directions

- Metadata are integral to every aspect of ATLAS computing
- The intent of this presentation has been to provide an illustrative view of ATLAS metadata, principally from the point of view of the infrastructure and services needed for metadata flow in the context of a single task
- While metadata components and infrastructure have grown organically as the experiment has matured, a number of principles described herein have informed their design and connectivity
- The infrastructure continues to evolve in a variety of ways, with improvements planned
  - to how dataset-level metadata may be used to reduce the need for peeking into input files,
  - to how metadata are emitted and transported from executing jobs to the collaboration's metadata repositories,
  - to machinery for robust accounting of low-rate error conditions in physics data bookkeeping