An inconvenient truth: file-level metadata and in-file metadata caching in the (file-agnostic) ATLAS event store

David Malon[1], Peter van Gemmeren[1], Richard Hawkings[2] and Arthur Schaffer[3]

[1]Argonne National Laboratory,
9700 South Cass Avenue, Argonne, IL 60439, USA
[2]European Organization for Nuclear Research,
CERN CH-1211 Genève 23, Switzerland
[3]LAL, Univ Paris-Sud, IN2P3/CNRS, Orsay, France

E-mail: malon@anl.gov

Abstract. In the ATLAS event store, files are sometimes "an inconvenient truth." From the point of view of the ATLAS distributed data management system, files are too small—datasets are the units of interest. From the point of view of the ATLAS event store architecture, files are simply a physical clustering optimization: the units of interest are event collections—sets of events that satisfy common conditions or selection predicates—and such collections may or may not have been accumulated into files that contain those events and no others. It is nonetheless important to maintain file-level metadata, and to cache metadata in event data files. When such metadata may or may not be present in files, or when values may have been updated after files are written and replicated, a clear and transparent model for metadata retrieval from the file itself or from remote databases is required. In this paper we describe how ATLAS reconciles its file and non-file paradigms, the machinery for associating metadata with files and event collections, and the infrastructure for metadata propagation from input to output for provenance record management and related purposes.

1. Introduction

High energy physics experiments have often found it useful to store metadata in event data files. Run records provide a canonical example. It is instructive, though, to consider the nature of such metadata, and the relationship of that metadata to the files into which they are placed. A number of ambiguities make elaboration of a definitive taxonomy of such metadata difficult, and as such a taxonomy is not central to a description of the ATLAS metadata framework, we will content ourselves with a description of various kinds of metadata arising in event file contexts.

Files are units of storage organization, not physics constructs, and while experiments endeavor to organize event data into files in a manner that aligns sensibly with data taking or physics use cases, very little "file-level" metadata really have any inherent relationship to files.

Some metadata, for example, pertain more properly to event collections. The physics metadata associated with events passing any of a certain set of triggers in a given run would be the same whether or not those events had been written into a file or set of files containing no other events. A list of pointers to those events within a larger sample would have exactly the same associated metadata, and an experiment's metadata infrastructure should provide a means to manage such

metadata in a manner that does not depend upon its file use. A trigger configuration may apply to a run, a geometry version, to a range of simulation runs, and so on, so that, even when such information is relevant to the events contained in a given file, the information pertains inherently to a set of events defined by physics or data-taking conditions, not by storage organization, a set that is often larger than and sometimes orthogonal to the given file's contents.

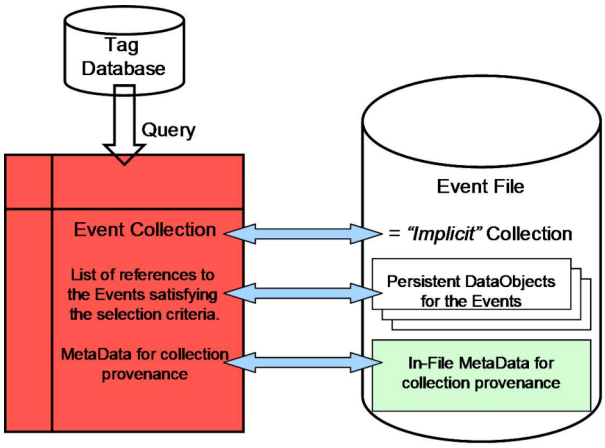

Figure 1: Comparing event collections and event data files ("implicit" collections)

Among the varieties of metadata we consider as in-file candidates are:

- Metadata inherently describing the sample contained in a file (see Figure 1 for correspondences between event collections and event data files). Provenance information (the run range and triggers from which the sample was selected, or the query that produced it) is in this category. Such metadata typically must be propagated to downstream data products, from raw trigger streams through reconstruction to analysis samples built by selecting, combining, and filtering events from many runs and possible trigger configurations, and beyond–through each successive stage in the analysis chain.
- Metadata describing the file's contents and how they were produced. The versions of the software components used to write the file, version and configuration information of the technology used in writing, the number of events and the processing stage (e.g., event summary data) are examples. Such metadata might not be propagated to downstream data products.
- Metadata cached in the file for convenience, to avoid remote lookups and database retrievals. Trigger configurations are a natural example. Such data often have an associated interval of validity (more on this later), and benefit from additional interval-of-validity-based retrieval machinery above the in-file metadata cache. Detector status information is another candidate for in-file caching, but there is a risk: detector status assessments may be updated long after event data files have been written, so in-file information may become stale and out of date.

Tellingly, many kinds of metadata most clearly associated with a file (size and checksum, for example) are not known until after the file has been written, and cannot be straightforwardly be written inside the file, or are mutable (e.g., ownership and access rights) and should not be recorded therein.

We expect that metadata inherently describing the file's production and the sample it contains are naturally stored within the file. For the potentially wide range of metadata that may be cached in the file for convenience, our guidelines for inclusion are based upon frequency of access and mutability: data that are needed by most clients, that are needed at multiple stages of analysis, and that are

unlikely to have changed since the file was written are the strongest candidates for inclusion in an in-file metadata cache.

2. A simplified view of the metadata output model

The ATLAS output model [1] is constrained by strictures imposed by the Athena/Gaudi control framework. In this framework, writing occurs via an output stream ("outstream") that is invoked once when the framework is initialized (via an initialize() method), once during each iteration of the primary event loop (via an execute() method), and once at the end of event loop processing (via a finalize()) method). The outstream is configured by listing the objects to be written (the "item list") and the transient store in which they reside, and by associating a specific persistence technology, along with any configuration details appropriate to that technology (the output file name in the case of file I/O, for example).

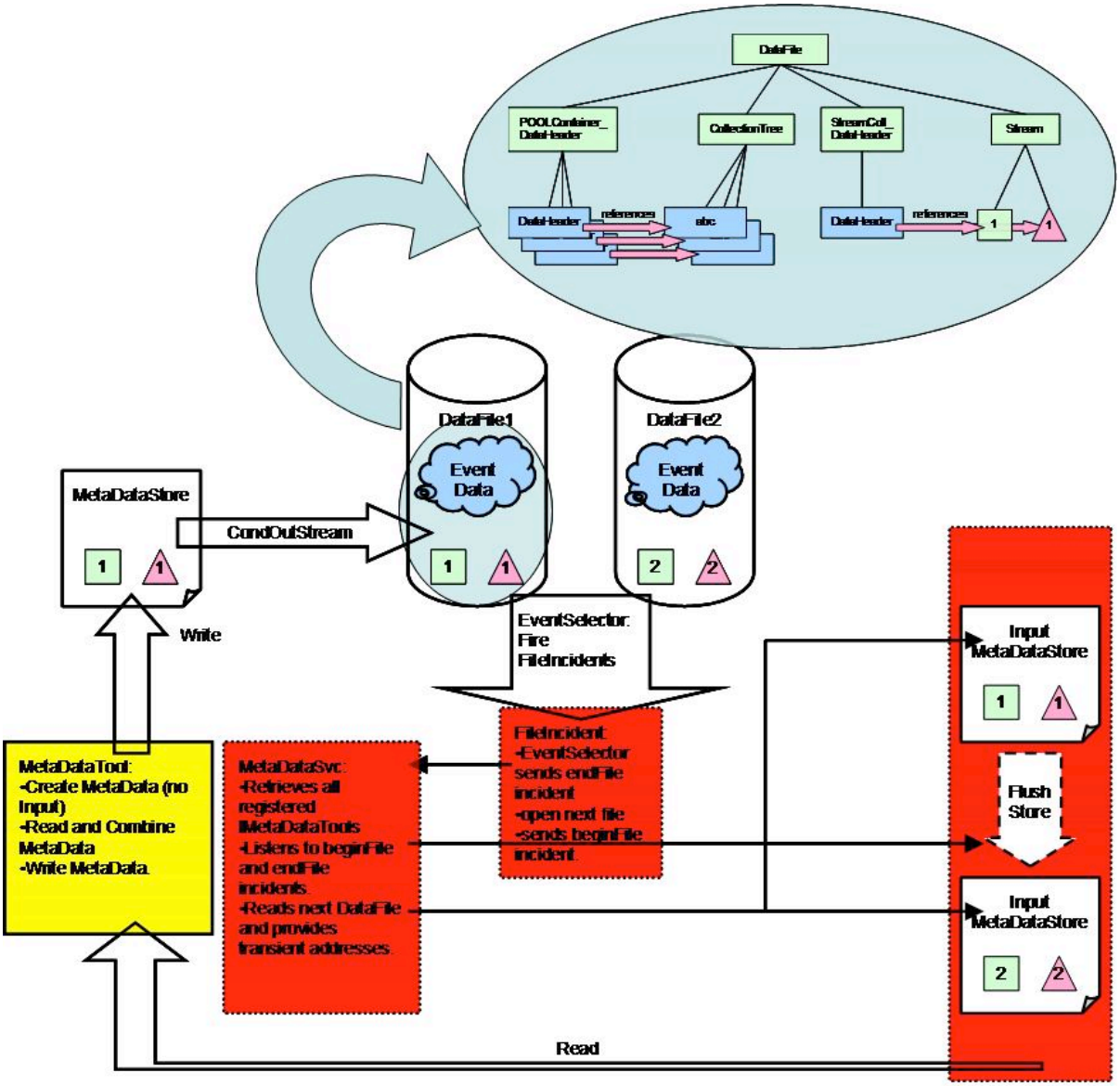

Figure 2: Overview of the file-level metadata framework

Figure 2 illustrates the framework for writing and reading file-level metadata. The opportunities to write metadata objects to a file occur most naturally, therefore, at the standard outstream invocation times. For the metadata use cases currently supported by ATLAS, it suffices to write metadata objects at finalization. More asynchronous writing may be accomplished by use of the control framework's incident services.

Metadata objects typically reside in a transient metadata store specifically instantiated to house them. They cannot, in general, reside in the transient event store, because that store is routinely cleared at the end of each event loop iteration, and most metadata objects have a longer lifetime or range of validity, or accumulate data for a range of processed events.

The item list approach makes output metadata extensible—anyone can create an additional metadata object and add it to the output item list.

3. Shadow streams

ATLAS writes event data and metadata into the same file by instantiating a "shadow stream"—an additional output stream that mirrors the configuration of the event output stream (file name, etc.), but with a different item list coming from a different transient store, and a "write on finalize" rather than a "write after each event" policy. A job that writes N event data streams might therefore have 2N total outstreams. ATLAS provides configuration tools that can make the creation and configuration of shadow streams all but transparent to users.

Outstreams are lightweight objects. In general, one could associate multiple metadata streams with a single event data output stream. Such a strategy might be employed, for example, to write a variety of metadata objects with different granularities and write control constraints.

4. In-file data organization

Whenever an Athena job writes the objects in an item list to persistent storage, it also writes a "data header" object—an object that tracks the transient names of those objects and the persistent locations at which they have been stored. For event data, the data header serves as a primary event entry point. (It does more—it also tracks the persistent locations of the data headers of the upstream input events, for example—but the details are beyond the scope of this paper.) The data header serves a similar purpose for metadata objects.

Because a data header is, by design, indifferent to the varieties of data to which it might point, event data headers and metadata data headers are indistinguishable by type. For this reason, care must be taken to separate them in file-based technologies that organize data by type. In POOL/ROOT, ATLAS creates a container for metadata objects that is separate and clearly distinguished from event data containers, under the configuration control of the shadow stream.

5. Metadata accumulation

Some varieties of metadata are accumulated by instrumentation of the output event stream. Statistics of various kinds are natural examples. The ATLAS I/O framework provides an extensible infrastructure, including output stream tools and a means to register them for invocation either before and/or after each write() operation, to allow accumulation of such metadata. A metadata object of this kind might accumulate values via the pre- or post-write calls, and register the accumulator object in the transient metadata store, for eventual writing with other metadata objects upon finalization.

Attaching such a metadata accumulator to a specific outstream as a tool rather than implementing it as an algorithm invoked for every iteration of the event loop eases collection of information specific to the given event data stream—an important distinction for jobs that write events to one (or more) of many possible output streams.

6. Metadata propagation

Several kinds of metadata should be propagated from input to output. Provenance metadata (e.g., the range of runs and triggers from which the sample was selected, needed for eventual cross section calculation) are a standard use case, but there are others. If one caches trigger configurations as in-file metadata, for example, then downstream data products may also need those configurations.

While copying metadata objects from input files or collections to output might be straightforward in some cases, in others, type-specific operations might be appropriate. For trigger configurations, for example, a set-like container might suffice, ensuring removal of duplicates when events from multiple input files share a trigger configuration. In the case of run or {run, luminosity block} ranges, slightly more complicated range mergers are required. (Note: a luminosity block is, loosely, a contiguous time interval within a run during which the integrated deadtime- and prescale-corrected luminosity may be determined.) The ATLAS metadata framework supports metadata propagation and extensible inclusion of type-specific metadata tools in the following manner.

The framework provides a metadata service ("MetaDataSvc") with which metadata tools may register in order to be initialized and added to the list of listeners for "begin input file" incidents. When an input file is opened, the MetaDataSvc retrieves the metadata data header and, from it, the addresses of any in-file metadata objects, making them available for retrieval in a transient input metadata store. When a "begin input file" incident is fired by the EventSelector upon opening a new input file, the metadata tools are thereby able to read their input metadata objects, process them or merge them as necessary with any metadata retrieved from earlier input files, and record the resulting, possibly extended, metadata objects in the (output) metadata store for later writing. (An "end input file" incident is also thrown after processing the last event in an input file, to alert any subscribers that the current contents of the input metadata store are about to be cleared and replaced with new data from the next file.)

7. Interval-of-validity-based retrieval

The preceding sections have described the basic infrastructure used by ATLAS to support in-file metadata. Many kinds of data, though, have a natural interval of validity (IOV), and an interval-of-validity-based retrieval infrastructure is widely used in ATLAS for consistent retrieval from conditions, calibration, and geometry databases. A geometry version might have an interval of validity that is a range of runs, a trigger configuration might have an interval of validity that is a single run, calibrations or Level 1 Trigger prescales might have an interval of validity that is (possibly discretized to) a range of luminosity blocks with in a run. ATLAS uses a temporal IOV database together with an interval-of-validity service within the Athena control framework to determine and load the correct version of a requested data type corresponding to a given event's timestamp or {run, luminosity block} numbers. The architecture supports separation of temporal database functionality from payload management—the time-varying conditions, calibrations, or configurations may or may not reside in the same database that manages the interval of validity and version information.

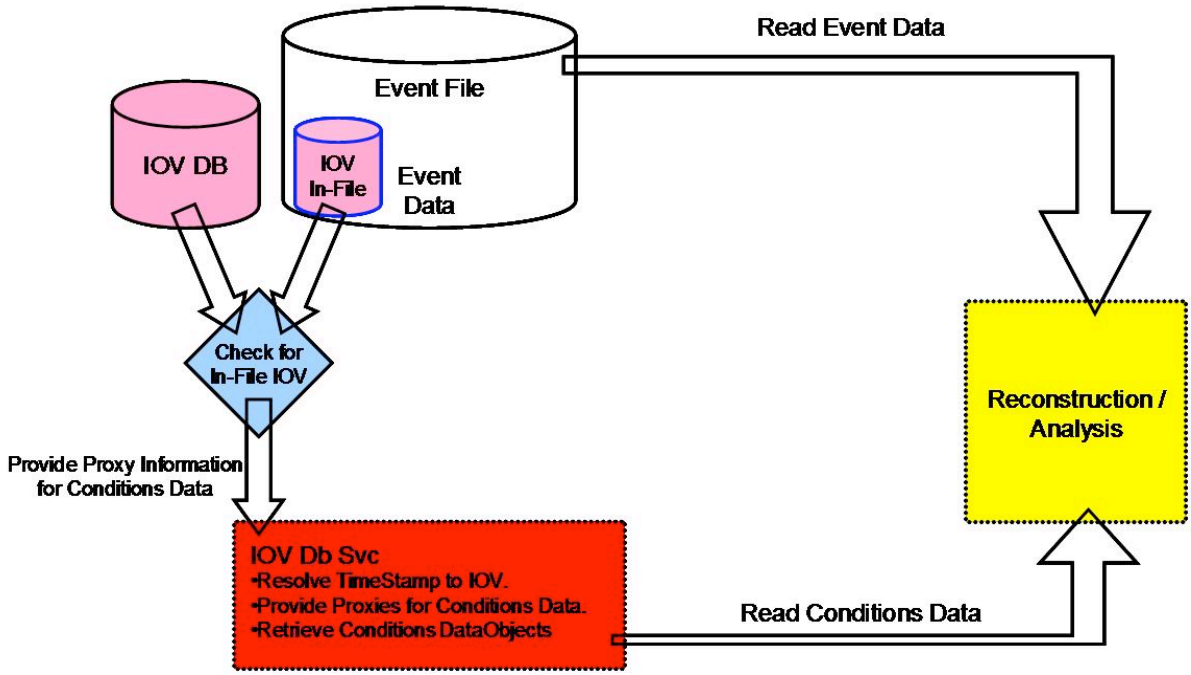

Figure 3: Using in-file metadata to robustly cache IOV data

ATLAS has extended its IOV-based retrieval infrastructure to support in-file metadata transparently, including in-file caching of interval of validity and version information (i.e., not just in-file caching of IOV-managed payload). From the point of view of the application, there is no difference between retrieval of IOV-based data from the event file or from a remote database or external file. In a standard configuration, a datum that cannot be found in the file is retrieved from a remote source as a fallback, though one may instead configure a job so that the remote source has primacy.

8. Conclusion

ATLAS has successfully developed a framework for in-file hosting of metadata in support of a varied and extensible range of information types and use cases, along with tools for intelligent metadata propagation and interval-of-validity-based retrieval. The volume and variety of metadata using this infrastructure is growing; indeed, the next challenge may be controlling the volume of metadata, using the adjudicating principles described in this paper.

9. References

[1] D. Malon, P. van Gemmeren, A. Schaffer, Sailing the Petabyte Sea: Navigational Infrastructure in the ATLAS Event Store *Computing in High Energy and Nuclear Physics (CHEP-2006)* **1** 312-315

[2] D. Costanzo et al, Metadata for ATLAS, *ATL-COM-GEN-2007-001*, 04 April 2007

Acknowledgments

Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under contract DE-AC02-06CH11357.