SAILING THE PETABYTE SEA: NAVIGATIONAL INFRASTRUCTURE IN THE ATLAS EVENT STORE

D. Malon, J. Cranshaw, P. van Gemmeren, Argonne National Laboratory, Argonne, IL 60439, USA
A. Schaffer, LAL et Univ. de Paris-Sud 11, Orsay, France

Abstract

ATLAS [1] has deployed an inter-object association infrastructure that allows the experiment to track at the object level what data have been written and where, and to assign both object-level and process-level labels to identify data objects for later retrieval. This infrastructure provides the foundation for opportunistic run-time navigation to upstream data, and in principle supports both dynamic determination of what data objects are reachable and controlled-scope retrieval. This infrastructure is complementary to the coarser-grained bookkeeping and provenance management system used to identify which datasets were input to the production of which derived datasets, adding the capability to determine and locate the objects used to produce specific derived objects. It also simplifies the task of populating an event-level metadata system capable of returning references to events at any stage of processing. The tension between what the infrastructure can demonstrably support at site-level scales--it is already extensively utilized by ATLAS physicists--and what is expected to be constrained by policy in light of anticipated distributed storage resource limitations is also discussed.

INTRODUCTION

Event Store Technology

ATLAS uses the LHC POOL [2] framework to persistify event data. While events may in principle be stored as objects in files or in relational databases, ATLAS currently uses only file-based storage for event data. Files have a unique identifier (GUID) used by POOL to support inter-object references that may cross file boundaries. File catalogs provide GUID-to-filename translation, and may support identification and location of multiple or alternate copies of the original file. POOL inter-object references contain sufficient additional information to locate objects within a file (e.g., a container name and an index into the container); further, to support typed references to objects, POOL makes use of a class identifier (CLID) that is unique for each class.

POOL's inter-object reference infrastructure provides a concrete foundation for implementation of the ATLAS event store navigational framework. How this infrastructure is integrated into ATLAS-specific transient object identification, into the control framework object retrieval architecture, and into ATLAS event store organization is described in the following sections.

Event Store Organization: Processing Stages

The ATLAS physics event store comprises a number of successively derived event representations, beginning with raw or simulated data and progressing through reconstruction into streamlined event representations suitable for analysis. Events arrive from the Event Filter (EF) in "bytestream" format.

RAW data are events as delivered by the Event Filter for reconstruction. Current estimates project an event size of 1.6 megabytes, arriving at a rate of 200 hertz. With the expected duty cycle of the ATLAS detector and the LHC, ATLAS anticipates recording more than three petabytes of RAW data per year.

Event Summary Data (ESD) refers to event data written as the output of the reconstruction process. ESD content is intended to suffice to render access to RAW data unnecessary for most physics applications other than calibration or re-reconstruction. ESD are stored in POOL ROOT files. A current size estimate is 500 kilobytes per event.

Analysis Object Data (AOD) provide a reduced event representation, derived from ESD, suitable for physics analysis. It contains physics objects and other elements of analysis interest. AOD are stored in POOL ROOT files. Current size estimates are 100 kilobytes per event. AOD data will also be divided into streams to facilitate creation of physics-group-specific datasets.

Event tags (TAG) are event-level metadata: event attributes chosen to support efficient identification and selection of events of interest to a given analysis. To facilitate queries for event selection, TAG data are stored in a relational database. Current size estimates are 1 kilobyte per event.

Simulation adds a number of processing stages to the sequence described above, including physics event generation (yielding so-called Monte Carlo truth), detector simulation (yielding detector "hits"), and digitization (yielding "digits" or detector readouts). A pileup stage (superposing data from other collisions in a time interval around the central event) may be part of digitization, or may be a separate processing stage. The eventual output of the simulation chain is written in Raw Data Object (RDO) format, and is stored in POOL ROOT files.

Access Patterns and Processing Stages

The decreasing size of event representations in successive processing stages allows a physicist to analyze a much larger number of AOD events, for example, than

ESD or RAW events. In addition, the AOD is likely to be more widely accessible, with a full copy at every Tier 1 and Tier 2 site. To further expedite analysis, a physicist beginning with a sample containing a very large number of events can issue a query against the TAG data to select a smaller subset of events for processing.

This increased agility at later processing stages comes at a price: difficult decisions must be made by the collaboration regarding what data appear in each stage. Not every use case will be satisfied by the contents of the AOD, for example, and someone who needs "just one more object" in the AOD must either work from the more unwieldy ESD, which is hosted by only a few large sites with less opportunistic, more production-oriented, access policies, or she must burden the rest of the collaboration with a larger AOD and greater storage resource costs. Equivalent issues arise in deciding the content of ESD vis-à-vis RAW, and so on. Sometimes, too, the access to "upstream" objects may be needed only for some events, so copying the data into downstream data products may be quite wasteful of resources.

It is for reasons such as these that the ATLAS event store supports a rich navigational infrastructure, one capable of supporting on-demand access by executing programs to data produced at any upstream processing stage, whether or not such data were specified explicitly as input.

EVENT PROCESSING

The ATLAS analysis control framework—Athena—is an implementation of the GAUDI [3] architecture. The ATLAS software architecture belongs to the blackboard family: data objects produced by an algorithm (e.g., a computational component of a reconstruction or analysis step) are given a name, or "key," and recorded in a common in-memory store ("StoreGate"). From this store other modules can access them by using the object's class ID and the string key. This model greatly reduces the coupling between algorithms for analysis and reconstruction (an algorithm does not need to know which specific module can produce the information it needs, nor which protocol it must use to obtain it). In ATLAS, this architecture is not hierarchical, so there is no common root object that could serve as an event entry point—the blackboard could, in principle, contain an assortment of completely unrelated objects.

Data Headers

In the absence of any required overall event data organization in the transient store, writing an event simply means persistifying a list of data objects. It is at this point that the ATLAS I/O infrastructure introduces a component called a DataHeader that may serve as a generic entry point for access to event data. The model is this: as each data object is persistified, its location in the persistent store is recorded, along with its transient identification, in the DataHeader. A DataHeader so constructed can therefore serve as an entry point or root object when the event is used as input to subsequent processing.

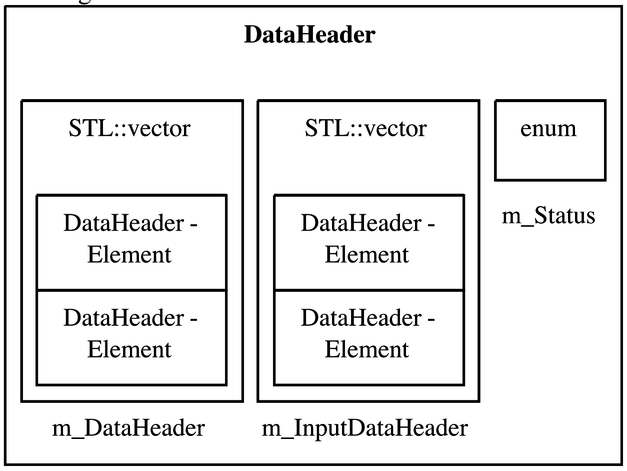

Figure 1: Layout of DataHeader.

The DataHeader (see Figure 1) stores a vector of references (called DataHeaderElements) to all data objects in the event. The DataHeader, in addition, reuses the DataHeaderElement construct to store a vector of references to upstream DataHeaders—and to itself—to facilitate back navigation.

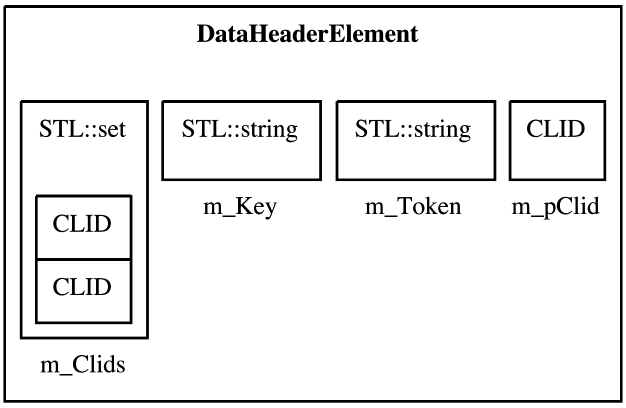

Figure 2: Layout of DataHeaderElement class.

The DataHeaderElement (see Figure 2) stores, in string format, the POOL pointer [4] to the referenced data object, and, as object identification, the object's class ID and key. (Reality is always more complicated. To support polymorphic data retrieval, multiple class Ids must in some cases be retained.)

Writing Data

When an event is written, an empty DataHeader is created. As each event data object is persistified, a DataHeaderElement referring to that object is added to the DataHeader. In addition to the resulting vector of references to all the data objects, the DataHeader stores another vector—the "provenance" vector—of DataHeaderElements referring to upstream DataHeaders, i.e., to event entry points for upstream processing stages.

This vector is filled by copying the provenance vector from the DataHeader of the current input event, and appending to this vector a self-reference to the current DataHeader. Because the output DataHeader is created as part of the output operation and does not have a transient store identity, the key used in the corresponding self-referential DataHeaderElement is, in principle, arbitrary. ATLAS uses this otherwise free field to declare a process tag name, such as RDO or ESD or RAW, to facilitate later scope-based navigation.

Sometimes a processing step merely copies input events to the output: this may happen, for example, when a physicist is extracting a subset of events for later analysis, or when many small files are being merged into larger files. In such a case, a simplistic approach to event-level provenance tracking would be needlessly expensive, causing back navigation to go, for example, from an AOD event to an earlier replica of exactly the same AOD event before reaching ESD. For this reason, a special "replacement mode" is provided, allowing DataHeaders to be written without extending the provenance vector, instead replacing the reference to the input DataHeader with a reference to the (equivalent) output DataHeader. This omission will cause the input file to be skipped during any subsequent sequential back navigation from the output of this processing stage.

Reading Input Events

A component called the EventSelector accesses events by iterating over the DataHeaders contained in each input POOL file, or over a list of references to DataHeaders when the input is not a list of files, but rather a list of events such as might be returned by a query to a tag database. The information in an event DataHeader's vector of DataHeaderElements referring to event data objects is used by a loadAddresses() function to pre-load proxies for all the data objects in the current file into StoreGate [5]. These data proxies allow the transient object retrieval infrastructure to determine quickly what data objects are available as input, and where to find them.

When an object is retrieved from StoreGate, the data are read from the input file and the object is created. If no valid proxy for the data object exists, then the data object was not among those available from the current input file.

This can happen because the analysis mistakenly tries to retrieve a non-existent data object (e.g., the key is misspelled, or the data object was not stored for the particular event) or because the data object was only written as part of an earlier upstream processing stage. To support the latter scenario, back navigation capabilities have been provided, allowing on-demand retrieval, for any or all events, of selected data objects from upstream data files. When this occurs, StoreGate will call an updateAddress() function in the EventSelector to attempt to update the proxy to contain sufficient information to support successful retrieval. It is at this point that back navigation machinery begins to play a role.

INTER-OBJECT NAVIGATION

During program execution, associations between data objects (e.g. between an electron object and a track object) can be made using C++ pointers, but such pointers have no meaning after the program terminates. Inter-object relationships are therefore implemented in the ATLAS transient data model using special reference classes known as DataLinks and ElementLinks. These classes manage associations by tracking the transient identification (class ID and key) of pointed-to objects, and, in the case of container objects, by tracking an index into them as well. Following such a link triggers the same object retrieval machinery as is triggered by a direct attempt by a physicist's algorithm to access an object in the transient store.

BACK NAVIGATION

Paddling Upstream: Sequential Back Navigation

In sequential back navigation, the updateAddress() function in the EventSelector will retrieve the upstream DataHeaders in reverse processing order (last in first out, e.g., AOD, ESD, RDO). After an upstream DataHeader is retrieved, its vector of DataHeaderElements referring to event data objects is searched to find the requested object. If no DataHeaderElement refers to the requested data object, than the next earlier upstream DataHeader will be retrieved and searched for a reference to the requested data object. The process is repeated until the first (most upstream) event DataHeader is reached or until a reference to the requested data object is found.

Retrieval will fail if the requested data object is not found in any upstream file or if an upstream file is inaccessible before the data object is found. If a reference to the requested data object is found in an upstream file, the data object is retrieved and stored in StoreGate.

There are times when back navigation is not desired—when, for example, one intends to read only AOD, one would not want to trigger a search of many files simply because of a typographical error in an object key. For this reason, a Boolean property in the job configuration controls whether back navigation will or will not be invoked.

Charting a Direct Course: Scope-based Back Navigation

A finer granularity and better performance for back navigation can be achieved by using the process stage information in the DataHeader. A physicist reading AOD, for example, may know that a data object is either available immediately upstream in the ESD or not at all. The physicist may also be able to provide retrieval hints: a "truth" particle may be stored only in the first processing stage (event generation), so it may be pointless to navigate through all the intermediate processing stages (and quite possibly to do this for each event) to reach the object of interest. In practice, this physicist may have copied only the generator event and the AOD to her home institute—after all, that is all she needs—so that navigating through the intermediate data stages may not only be expensive, it may be impossible. ATLAS is currently implementing scope-based back navigation, allowing the user to restrict or direct the scope of back navigation to specific named processing stages.

Implicit Back Navigation

Inter-object links (as described earlier) can also be used for inter-file references. Whether a pointed-to object is in the same or a different file is transparent to the client. A photon in the AOD, for example, may contain a link to a calorimeter shower in the ESD. If back navigation is enabled, following a DataLink or ElementLink to a data object belonging to an upstream processing stage will cause a StoreGate retrieval to trigger the back navigation machinery described above.

EVENT TAGS AND NAVIGATION

The event tags contain event-level metadata useful for event selection. They are derivable from data contained in the AOD, and are in practice written when AOD are written, or when small AOD files are merged into larger ones. The event tags therefore contain a reference to the associated AOD DataHeader, but they can also record any or all of the references to upstream data available in the AOD DataHeader's provenance vector. This allows one to query the tag database and get a list of references to any stage of processing for that event. This can be useful in a variety of ways. For example, if one needs to re-run reconstruction on raw data, but only for selected events, then one could use the tag database for event selection and return references to the RAW data for qualifying events, thereby avoiding running over the entire event sample. If a physicist knows in advance that her jobs will need objects from the ESD as well as the AOD, the event tags can give return a list of all files (or file pairs) needed. This file list is in fact a list of file ids (GUIDs), so an additional file catalog lookup is required to locate the corresponding files. Results of queries are routinely grouped by file GUID for efficient processing, and to support parallelization. Utilities to extract the list of unique GUIDs corresponding to an event selection are also provided, and may be used by data transfer tools or resource brokers.

DISTRIBUTED ANALYSIS

ATLAS navigational infrastructure has been designed to work in either local or distributed environments. When a user attempts to retrieve an object for which back navigation would be required, it may happen that a file containing the object is identifiable in the catalogs that are consulted, but that all replicas of the file are remote. Whether remote data access or file transfer is triggered at this point is a matter of site policy, not a matter of design limitation. (In practice, it is expected that many ATLAS sites will allow run-time navigation only to onsite data.)

OUTLOOK

Back navigation capabilities were deployed in advance of the 2004-2005 ATLAS Data Challenge, and were widely employed in work leading up to a June 2005 ATLAS physics workshop. The machinery proved popular, efficient, effective, and useful—so much so that, perhaps paradoxically, ATLAS users have been cautioned not to rely on it too much: sites that will be able to host the ATLAS ESD may not be prepared to deal with a large number of users attempting opportunistic back navigation to such data. Commissioning exercises in 2006 will provide important feedback regarding how ATLAS event data products are (or should be) partitioned, and regarding the requirements for access to and dynamic navigation between event data in its various stages and incarnations.

ACKNOWLEDGEMENTS

The submitted manuscript has been created by The University of Chicago as Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. W-31-109-Eng-38. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

REFERENCES

[1] W.W. Armstrong et al., "ATLAS Technical Proposal", CERN/LHCC/94-43.

[2] D. Duellmann et al., "POOL Project Overview", CHEP 2003: Proceedings, eConf C0303241, MOKT007 see also: http://pool.cern.ch

[3] M. Cattaneo et al., "Status of the GAUDI event-processing framework", CHEP 2001: Proceedings. Edited by H.S. Chen. Beijing, China, Science Press, 2001. 757p.

[4] D. Duellmann et al., "The POOL Data Storage, Cache and Conversion Mechanism", CHEP 2003: Proceedings, eConf C0303241, MOKT008

[5] P. Calafiura et al., "The StoreGate: a Data Model for the Atlas Software Architecture", CHEP 2003: Proceedings, eConf C0303241, MOJT008.