BEYOND PERSISTENCE: DEVELOPMENTS AND DIRECTIONS IN ATLAS DATA MANAGEMENT

D. Malon*, Argonne National Laboratory, Argonne, IL 60439, USA
A. Schaffer, LAL, Orsay, France

Abstract

As ATLAS begins validation of its computing model in 2004, requirements imposed upon ATLAS data management software move well beyond simple persistence, and beyond the "read a file, write a file" operational model that has sufficed for most simulation production. New functionality is required to support the ATLAS Tier 0 model, and to support deployment in a globally distributed environment in which the preponderance of computing resources—not only CPU cycles but data services as well—reside outside the host laboratory. This paper takes an architectural perspective in describing new developments in ATLAS data management software, including the ATLAS event-level metadata system and related infrastructure, and the mediation services that allow one to distinguish writing from registration and selection from retrieval, in a manner that is consistent both for event data and for time-varying conditions. The ever-broader role of databases and catalogs, and issues related to the distributed deployment thereof, are also addressed.

INTRODUCTION

ATLAS data management software has been deployed at multi-terabyte scales in recent years, but a 2004 data challenge (Data Challenge 2) provides the first real test of many of its capabilities. In earlier deployments, it sufficed to support a "file in, file out" model of processing: generate files of events, simulate them, superpose pileup events, digitize them, reconstruct them, and analyze them. The basic processing pattern was to iterate over the input events in a file, process them, and write the results into a new file. In Data Challenge 2, ATLAS will mix event streams to produce samples that resemble what is expected to come off of the detector via the high level trigger, and will exercise the processing chain planned for the Tier 0 center at CERN to build Event Summary Data (ESD), Analysis Object Data (AOD), and TAG databases to support event selection. These data will be distributed to remote sites for analysis in a manner consistent with the proposed ATLAS computing model.

New (or newly utilized) capabilities include support for multiple, possibly overlapping output streams, support for collection building using registration services, and back navigation with policy control.

*Work supported in part by the US Department of Energy, Division of High Energy Physics, under Contract W-31-109-ENG-38

A great deal of effort has gone into articulation and refinement of the ATLAS event model and into its persistification, which is still limited by the capabilities of the primary technologies upon which the ATLAS event store will be based: the LHC Computing Grid (LCG) Project's common persistence infrastructure POOL [1], and the ROOT [2] software upon which POOL's file-based storage relies. The associated difficulties in persistification and how they have been addressed are beyond the scope and size limitations of this paper.

STREAMS AND COLLECTIONS

First-pass reconstruction processing will divide output events into multiple streams. While the database software imposes no restrictions on the number of streams, on whether streams may overlap, and on stream-specific specification of event data content, the Data Challenge 2 model is that the numbers of streams is small (~10), that streams are disjoint, and that streams share the same content definition. This choice may be revisited after the data challenge, but the rationale is that unless one can provide reassurance that no one in the ~2000-person collaboration will want to analyze samples that cross stream boundaries, this is the safest strategy: otherwise, a cross-stream sample would find certain data available for some events and not for others, depending upon the stream from which the events were extracted, and one would need safeguards against the very real possibility of processing the same event multiple times.

In this model, streams serve principally as physical clustering optimizations—they are not necessarily the units of input to physics analyses. The latter is the role of collections. The idea is that, while first-pass reconstruction will write events exactly once, references to those events will be recorded in as many physics collections (event lists, really) as might want them, along with associated metadata ("tag" data) to support more detailed event selection. Higgs candidate events, for example, might not get their own stream, but they would almost certainly get their own collection; moreover, one could imagine two very similar samples, differing only in their cuts, being instantiated as two different collections, but not as two different streams.

WRITING AND REGISTRATION

The distinction between writing and registration is an architectural motif that is increasing in prominence in the ATLAS data management framework: there is a difference between storing an object and remembering where you put it. When phrased this way, the distinction may appear obvious, but a great deal of work may be done—and indeed has been done by many experiments—without ever formalizing the difference. In simulation production and analysis, for example, one reads all the events in a generator file and writes all the events into a hits file, or reads a file of digitized events and writes a file of reconstructed events. Except for remembering the file names, one does not need to record where individual events have been stored. The only registration happens at the file cataloguing level.

The same phenomenon may be observed in many conditions database products [3, 4], in which one often cannot write a conditions object without assigning it an interval of validity. An ATLAS design contribution (and an ATLAS requirement) to the LCG common project on conditions data management has been the architectural separation of payload storage from registration in a temporal database.

The ATLAS control framework (Athena) endeavors to provide a consistent view of writing and registration processes for all kinds of data. In Athena, "outstreams" are used to control the writing of data objects. When an object is written, a token, opaque to the user, is returned. On input, this token is used by persistence services to locate and return the object of interest.

The ATLAS control framework supports the notion of registration services, wherein one can record an object's token, along with optional metadata that may later be used to help decide which objects are of interest. The registration model is the same for event and non-event data: only the concrete type of the registration service changes. For event data, event collections, currently based upon POOL collections, provide the repository in which registration information is retained. This is how ATLAS tag databases are constructed: users record references to events, along with metadata that might be used for later event selection. For conditions and calibration data, a temporal database serves as the registration service, recording references to the data, along with associated metadata—in this case, an interval of validity, a folder name (used to organize conditions data types), a tag, and so on—but the registration motif remains the same. (As an implementation optimization, one may choose to store conditions data on the same server, and even in the same database, as the temporal database that mediates access, but the architectural distinction between writing and registration, and between payload and metadata, remains.)

COLLECTIONS IN PRACTICE

If the collections model is successful, event collections will permeate the ATLAS analysis environment. A simplified scenario for event selection might be:

- Query collection-level metadata in a collection or dataset catalog to identify a collection of interest;
- Apply a filter predicate (query) to the collection to build a list of events of interest—the result is another collection;
- Extract the list of unique file ids from the resulting event list, to give to grid resource brokers for resource acquisition and scheduling;
- Move the collection of selected events (file-resident) into the job sandbox as input to the analysis—the job will iterate over these events and no others.

Note that this is an oversimplification, in that the resulting collection would, if large, be partitioned on file boundaries, and multiple jobs corresponding to the input partitioning would be submitted to analyze the sample, with an output concatenation step at the end. Work is in progress along these lines in the ATLAS Distributed Analysis (ADA) project.

For Data Challenge 2, ATLAS has deployed the utilities to extract into a file the references to qualifying events as the output of a query, and to build the list of unique file ids needed by a grid resource broker. Additional utilities to build tailored samples with specific content have also been delivered, e.g., to extract into a physicist's personal files the ESD, say, for all events that satisfy a given query to the TAG database.

BACK NAVIGATION

We use the term back navigation to refer generically to the machinery to support retrieval of data produced in earlier processing stages, e.g., the ability of a reader of AOD to retrieve data in ESD, or even RAW or MC TRUTH data objects. In the ATLAS framework, there are two mechanisms by which back navigation might be accomplished. The first is by following direct references between data objects. In this case, the store is agnostic to processing stages, and simply follows the persistent pointers, assuming that data are locatable. The second mechanism is by "name" (in ATLAS, this is by data type and user-assigned key): a physicist may attempt to retrieve by name from the transient store any object that was saved in earlier processing stages. Data lookup and delivery is by recursive delegation: the input event (AOD, say) is asked whether it can deliver an object with that name; if not, it asks its parent (ESD) the same question, and so on until the object is found or the upstream data stages are exhausted.

While the database software supports arbitrary back navigation, the framework also delivers "policy hooks" to allow control (including depth control) of back navigation. In the ATLAS computing model, it is likely that back navigation from AOD to ESD, while

theoretically possible anywhere, will be routinely supported at Tier 1 centers, which host both AOD and ESD, but will be unsupported or costly at sites that locally host only AOD (e.g., at Tier 2 centers and beyond).

A component strongly related to back navigation support is provenance management. Each persistified event retains a reference to its parent. When one produces ESD from RAW and AOD from ESD, the provenance is obvious, but what happens when RAW→ESD→AOD is accomplished in a single job via concatenation? Additional machinery is required to ensure in this case that ESD and AOD do not both believe they are the daughters of the RAW input event, and instead to ensure that AOD "knows" that ESD is not its sister, but its mother. .

ON SIMPLE PERSISTENCE

In the early stages of a software project, simple persistence is a boon. Developers appreciate the ability to write their objects (or, more accurately, their object states) and read them back later, without worrying about schema, transient-to-persistent conversion, and persistent data organization. Such a capability may speed development, but it comes at a cost: the persistent store, rather than having an explicit design, is built, *de facto*, of snapshots of the transient data model at the time that data producers finish execution, and a reader's view of the data must be the writer's view.

Object orientation in some ways exacerbates this potential problem. With ntuples, one can in principle retrieve only the attributes of interest to an analysis, whereas object persistence mechanisms in general allow one only to retrieve and rebuild in their entireties objects as they were written, not to gather selected (usually private) attributes from a variety of objects and reconstitute them into a new object.

Relational databases usually do better than this: one can write tables with a great many columns, and, via a query, extract exactly the columns one needs. With a typical object persistence infrastructure, one cannot write FullTracks and retrieve ParameterizedTracks; one must first retrieve sufficient data to rebuild FullTracks, and only then project them onto ParameterizedTracks.

Many developers expect that schema evolution will help them manage a collaboration's evolving view of its data, and it may. The LCG POOL project will deliver some schema evolution support in Release 2.0, based upon ROOT 4's schema evolution capabilities. Such capabilities may handle simple cases (addition or removal of a data member; data member type changes), but more substantial refactoring of an experiment's event model—even alterations as natural as adding association objects rather than relying upon direct pointers—will require substantially more than schema evolution is likely to provide.

An object persistence infrastructure tends to be ambiguous in many areas, especially those related to object identity, equivalence, and substitutability: If I write an object state, then copy it to a different location, is it the same object, is it different but substitutable for the original, and how do I find it? If I instead write the object twice (for example, to two different streams), are the two instances equivalent, and how do I know? Are these answers different if the state of the transient object changes between writes, and how is this managed?

BEYOND PERSISTENCE

To move beyond simple persistence, several developments are required. The ATLAS software supports separation of transient and persistent type identification, so that, in principle, one can deliver a system that, on input, asks, "Can I build a transient object of type A from the a persistent state object with shape B," rather than exclusively restoring As from As and Bs from Bs. The LCG POOL infrastructure, in response to ATLAS requirements recorded in the Requirements Technical Assessment Group (RTAG) report that served as the charter for POOL, provides hooks to make this task easier, but they have not been utilized by any experiment to date. Such capabilities will be essential to support reader's views that are not writer's views, and to support the kind of event store evolution that one must anticipate for long-lifetime experiments such as ATLAS.

A multi-petabyte event store will require a sophisticated metadata infrastructure and navigation machinery to ensure that data of interest can be efficiently identified, located, and accessed. ATLAS has begun to deploy such an infrastructure; its 2004 data challenge will provide early feedback regarding the viability of the proposed ATLAS event store design.

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] http://pool.cern.ch.
[2] http://root.cern.ch.
[3] http://lcgapp.cern.ch/project/CondDB.
[4] http://kdataserv.fis.fc.ul.pt/ATLAS/#CondDB.
[5] D. Malon, "What Your Next Experiment's Data Might Look Like: Event stores in the LHC Era," Proceedings of the Meeting of the Division of Particles and Fields of the American Physical Society (DPF 2004), Riverside, CA, August 2004, to appear.