The Athena Control Framework in Production, New Developments and Lessons Learned

C. Leggett, P. Calafiura, W. Lavrijsen, M. Marino, D. Quarrie, LBNL, Berkeley, CA 94720, USA

Abstract

Athena is the Atlas Control Framework, based on the common Gaudi architecture, originally developed by LHCb. In 2004 two major production efforts, the Data Challenge 2 and the Combined Test-beam reconstruction and analysis were structured as Athena applications. To support the production work we have added new features to both Athena and Gaudi: an "Interval of Validity" service to manage time-varying conditions and detector data; a History service, to manage the provenance information of each event data object; and a toolkit to simulate and analyse the overlay of multiple collisions during the detector sensitive time (pile-up). To support the analysis of simulated and test-beam data in Athena we have introduced a python-based scripting interface, based on the CERN LCG tools PyLCGDict, PyRoot and PyBus. The scripting interface allows to fully configure any Athena component, interactively browse and modify this configuration, as well as examine the content of any data object in the event or detector store.

ATHENA AND GAUDI

Athena is the object oriented control framework used by the ATLAS experiment at CERN, which is based on the Gaudi architecture. The Gaudi framework, originally developed by LHCb, is now also shared by ATLAS, GLAST, HARP and OPERA. It is written in C++, and is designed with a modular component architecture, consisting of a handful of core packages, such as the kernel, services and various tools, and supplemented by external libraries such as POOL, SEAL, PI and Geant.

The framework is designed to maintain a strict separation between transient and persistent data, where components code to abstract interfaces. This allows individual components to be easily replaced as technologies evolve, which is essential for an experiment which will run for several decades.

Athena comprises the ATLAS specific extensions to Gaudi, most notably

- StoreGate - the data store [1]
- Interval of Validity Service - managing time dependent data
- Pileup - combining multiple events in the detector
- History Service - maintaining a multi level record of data provenance
- Python scripting

NEW DEVELOPMENTS

Interval of Validity Service

The Interval of Validity Service (IOVSvc) makes associations between user data and time dependent data that resides in specialised conditions databases. It is designed to be transparent to the average user, with low overhead on the system. Two access patterns are offered: registration of a handle to user data with a specific entry in a conditions database, where the data is automatically updated where necessary, and registration of a callback function with a specific entry in the conditions database, where the callback function is triggered when the data enters a new validity region.

In order to minimise unwanted database access, the data is only read from the database when the handle is dereferenced.

Hierarchical registration is possible, so that it is possible to register handle B against a previously registered handle A, or function F2 against function F1. Callback functions are assembled into an acyclic graph, and are triggered in the appropriate sequence. It is possible to register a handle or function against multiple objects.

Both the validity information, and the time dependent data can be preloaded on a job or run basis, for specialised applications such as triggers or test beam setups, where continuous database access is unwanted.

Detector Pileup in DC II

For Data Challenge II, over 1000 minimum bias events are overlaid over the original physics stream. The main requirement was that the digitisation algorithms should run unchanged. Many optimisations of data structures and access patterns were necessary in order to allow pileup to run on a standard node, as original memory consumption was extravagant. Current memory requirements are < 1 GB.

The Tuple Event Iterator manages multiple input streams. Random permutations of events are selected from a circular buffer of minimum bias events. Since the various subdetectors have different data integration times, they require individual cache retention policies. By using a two-dimensional detector and time-dependent event caching policy, memory utilisation has been significantly reduced.

Pileup is an excellent mechanism to stress test the architecture. Small problems which would normally pass unnoticed, are enormously magnified, and become visible far sooner. It is also an excellent tool to expose memory leaks, as they become 1000 times larger.

History Service

It is essential that the provenance of data be assured. Users must be able to see the full history of processing and reconstruction, in order to make meaningful decisions on data quality and cuts, and to be able to reproduce calculations at a later data. The History Service keeps track of multiple levels of linked provenance information:

- job environment
- full job configuration
- Services instantiated
- instantiated Algorithms, AlgTools, and SubAlgorithms
- DataObjects

When a DataObject is recorded in the event store, a HistoryObject is created and associated with it, with the same retention policy. The HistoryObject contains links back to the HistoryObject of the Algorithm which created it, which links to the HistoryObject of the job.

The service is invisible to the user, but can be switched off for specialised situations such as the High Level Trigger, where it is unnecessary.

Python Scripting

Python has been woven throughout the Athena framework, being used both for scripting, and for interactive access to C++ objects. We have replaced the flat text based configuration files of Gaudi with files, which are executable Python fragments. This allows us to do dynamic job configuration with full access to Python functionality, such as variable manipulation and conditional branching. By using this system, it has become very easy to turn detectors and reconstruction or data processing tasks on and off in a two dimensional matrix.

Python has also provided us with the ability to do interactive analysis, allowing users to individually cycle through events and providing access to C++ objects from the Python prompt. It also allows histograms and ntuples to be generated and accessed in an interactive and dynamic manner.

ATHENA IN PRODUCTION

Data Challenge II

Atlas is currently in the midst of its second Data Challenge. It consists of:

- Phase 1:
  - Event generation: 30 Physics channels, 10s of millions of events
  - Detector simulation: using Geant 4, track particles through the detector and record interaction of particles with sensitive elements in the detector
  - Pileup and Digitisation: output bytestream raw data
  - Data transfer to CERN: 35 TB in 4 weeks
  - Event mixing: combine physics events in ad-hoc proportions
- Phase 2: reconstruction and real time distribution of data to Tier 1 institutes
- Phase 3: worldwide distributed analysis using the Grid

Phase 1 is currently running on 3 grids: LCG (Europe, Japan, FNAL and TRIUMF), NorduGrid, and Grid3 (US). The main difficulties encountered to date have been grid configuration, use of certificates, access to conditions database over the grid, and the debugging of production systems.

Combined Test Beam

Atlas has been recording data from its Combined Test Beam since July, with various detector configurations, during which time over a terabyte of data has been written from about 5000 runs. We are using evolving releases of Athena to make use of new features and fixes, such improvements to the multithreaded architecture, and advances in the conditions databases. The Geant4 simulation and reconstruction of the CTB setup are occurring in parallel, with continual comparison between the two. Conditions databases are now if production mode, being used for both reading and writing.

We are currently preparing for phase II of the Combined Test Beam plan, which involves a massive reconstruction effort of all real data and production of Monte Carlo data.

LESSONS LEARNED

In the course of putting Athena into production, we have become aware of several issues, few of which were particularly surprising. As always, requirements changed during the course of the development cycle, requiring several shifts in the design of the framework. In certain circumstances, such as for pileup, it was very important to design the architecture and access patterns with performance in mind. Furthermore, pileup proved to be an excellent test bed for stress testing the architecture in general.

Database access, especially over networks and on the grid, continues to be problematic. Where local testing ran unimpeded, when used in production mode many issues were encountered. This showed how spanning the gaps between component and integration tests is very often hard to accomplish.

Persistency is always a concern, and we have been forced to redesign our persistency mechanism several times. Despite attempts to maintain a clear separation between transient and persistent layers, certain objects and container classes had to be rewritten while paying close attention to the persistency mechanisms.

While running the Combined Test Beam, we have discovered that better support for realtime monitoring is essential. Python scripting has proved invaluable in this regard, being able to examine objects interactively, but work still needs to be done. Python in general has made dynamic job configuration much less painful.

Finally, while we are pleased with the current design and progress of Athena, we still need more feedback from users and developers to better understand if sufficient functionality has been built into the system.

ACKNOWLEDGEMENTS

We would like to thank all ATLAS collaborators who contributed to the design and prototyping of Athena, as well as the Gaudi developers on LHCb.

This work was supported in part by the Office of Science, High Energy Physics, U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

REFERENCES

[1] P. Calafiura, C. G. Leggett, D. R. Quarrie, H. Ma and S. Rajagopalan, eConf **C0303241** (2003) MOJT008 [arXiv:cs.se/0306089].

[2] 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.