Upgrade of ATLAS data quality monitoring for multi-threaded reconstruction

Tomasz Bold¹, Walter Lampl², Rohin Narayan³, Peter Onyisi^{3,*}, and Piotr Sarna¹, on behalf of the ATLAS Collaboration

¹Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, Krakow, Poland

²Department of Physics, University of Arizona, Tucson AZ, USA

³Department of Physics, University of Texas at Austin, Austin TX, USA

Abstract. ATLAS is embarking on a project to multithread its reconstruction software in time for use in Run 3 of the LHC. One component that must be migrated is the histogramming infrastructure used for data quality monitoring of the reconstructed data. This poses unique challenges due to its large memory footprint which forms a bottleneck for parallelization and the need to accommodate relatively inexperienced developers. We discuss plans for the upgraded framework.

1 Introduction

The ATLAS experiment [1] at the Large Hadron Collider (LHC) utilizes multiple layers of monitoring to verify the condition of the detector and the quality of collected data. The monitoring system includes mechanisms to produce monitoring information (primarily in the form of histograms), to provide visualizations for users, to perform automatic checks on the information, and to provide persistency for the results.

One core component, *AthenaMonitoring*, is code that runs in the ATLAS data processing framework Athena [2] (based on Gaudi [3]) to produce histograms as part of the standard reconstruction of the data. In preparation for Run 3 of the LHC, the Athena framework is being updated to support a multithreaded execution mode. This will require many changes to the Athena user code base, including *AthenaMonitoring*. This presents an opportunity to rectify some flaws in the current monitoring framework implementation.

2 Current Implementation of AthenaMonitoring

The Athena framework currently uses a single-threaded model in which a sequence of *algorithms* is executed on events in an order determined in a job initialization step driven by Python configuration files. Algorithms share information by writing and reading objects stored in a data “whiteboard.” The algorithms can execute *tools*, which can be configured independently of the algorithms. Algorithms can also interact with *services* which live outside the event loop; of particular relevance to monitoring is the histogramming service (THistSvc),

*e-mail: ponyisi@utexas.edu

Figure 1. Schematic of the execution of monitoring in a current Athena reconstruction job. A strict order of algorithms is established in the job configuration, in which monitoring algorithms are run after the data they use are produced by the reconstruction algorithms. The monitoring algorithms interact with the THistSvc service to store produced histograms into ROOT files.

Figure 2. Relationship between the components of the AthenaMonitoring framework; this reflects both the current framework and the anticipated evolution. Users subclass a base Athena tool class and add code to book histograms and fill them with information retrieved from the event data whiteboard. These tools are added to an instance of a generic Athena monitoring manager algorithm, which executes the tools when the algorithm is executed in the reconstruction sequence. Generally there will be several instances of the manager algorithm, each controlling related sets of user tools (for example, one algorithm per detector).

which handles the saving of ROOT [4] histograms to files. A schematic diagram is shown in Figure 1. Data quality monitoring algorithms should only consume information added by other algorithms, and should not modify the data whiteboard.

AthenaMonitoring is implemented as a generic Athena algorithm and a Athena tool base class which is then subclassed by user code. The algorithm is a very thin layer which primarily exists to provide a scheduling unit for the execution sequence. User code (in the form of a tool subclassing the common monitoring tool class) is responsible for the initial memory allocation of histograms, setting the binning and properties of the histograms, retrieving necessary information from the whiteboard, and filling the histograms. The superclass ensures that the histograms are placed in the appropriate output locations, are rebooked when necessary (to handle time-dependence for example), and that common trigger and bad event filters are applied.

A typical full reconstruction job in the 2018 configuration schedules around 35 top-level AthenaMonitoring algorithms covering 160 individual monitoring tools. These produce about 80 thousand histograms, using around 1 GB of memory.

The current division between the responsibilities of the framework and of the user code results in a number of undesirable consequences:

- the user code has access to the raw histogram pointers. This strongly limits how much management the central code can provide, as the user code can in principle do anything to any histogram at any point. Transitions to new technologies (for example, ROOT 6 to ROOT 7-style histograms) are completely dependent on user adoption, and the framework needs to be able to handle all potential user choices.
- histogram parameter declaration happens in user C++ code. Deploying minor changes, such as adjustments to histogram binning, requires a new release of Athena. In situations where run conditions are rapidly evolving, this can be a significant problem (for example, inappropriate binning resulting in most entries being marked as overflow).
- the semantics of ROOT memory management and storage differ in subtle ways for several of the most commonly-used object types (TH1, TGraph, TTree), requiring somewhat fragile code to handle. The interaction between central management code and user code can and does lead to subtle bugs unless users strictly adhere to programming contracts that differ by object type.

3 The Future: AthenaMT

Current trends in CPU design yield an increasing number of computing cores in a single system rather than increasing the clock speed of a single core, so taking full advantage of new systems requires the ability to run in parallel on many cores. The amount of memory in a typical system is not scaling at the same rate, leading to a reduction in the typical amount of available memory per core. Although collider event reconstruction is an embarrassingly parallel problem, the current single-threaded ATLAS event reconstruction is so memory-intensive that available memory can limit the number of simultaneous independent single-threaded tasks that can be run at the same time.

An initial solution to this problem, multiprocess Athena (AthenaMP) [5], uses process forking and the copy-on-write mechanism to run multiple independent processes, derived from a single initial process; memory pages that the daughter processes do not modify remain shared. This mechanism is currently deployed for reconstruction workflows run on the Grid [6] and results in significant memory savings. Unfortunately, AthenaMP does not help with monitoring code: histograms filled in reconstruction cannot be shared between processes, and their memory consumption is approximately linear in the number of processes. With only a few cores, this can already result in monitoring being the single largest memory user in an AthenaMP reconstruction job.

A further development, multithreaded Athena (AthenaMT) [7], will result in a single-process, multi-thread workflow. Since multiple cores will share the same memory space, memory consumption can in principle be independent of the number of cores. To permit multiple events to be processed simultaneously on independent threads, the Athena code must be made thread-safe. This will be done on a per-algorithm basis in one of several ways:

- the algorithm can be “legacy,” such that only one instance is available over all threads, and access is protected by a mutex. This can be applied to any code but reduces potential concurrency;

- the algorithm can be cloned, such that each thread has an independent copy with its own local state. This improves concurrency at the expense of memory use;
- the algorithm can be reentrant, such that multiple threads can execute the code of a single instance in parallel. This provides the best concurrency and memory use at the cost of requiring user code to be thread-safe.

Cloned monitoring algorithms have the same behavior as monitoring in AthenaMP (memory use linear in the number of threads), so this is not an acceptable solution. Monitoring algorithms must either be legacy or reentrant.

AthenaMT will not execute algorithms in a single fixed linear sequence, as this ignores potential intra-event parallelism. Instead, every algorithm will declare the data objects that it needs and which objects it will provide; a dependency graph built from this information will be used to determine which algorithms are available to be executed on the available data at any given time. This requires current Athena code, which does not declare this information, to be modified. As a result, all existing user monitoring code will have to be altered to some extent.

There are a large number of histograms produced in reconstruction monitoring that are typically expected to be empty or to be very sparsely filled, such as those representing error conditions. A special-purpose histogramming package, “Light Weight Histograms,” was developed internally in ATLAS in 2009 to handle sparse histograms in an efficient manner. This package aggressively uses global memory pools in a non-thread-safe way and altering the monitoring code to use a more standard cross-experiment solution is preferred over attempting to port the package to AthenaMT.

Strictly speaking, after modification to declare data dependencies, current monitoring code could generally run in AthenaMT jobs as legacy algorithms as defined above. Since there are a large number of independent monitoring algorithms which have no interdependencies and produce no output data objects, this would most likely lead to acceptable performance for a low number of threads. This is a fallback solution for user code that proves difficult to migrate to the new framework.

4 The New AthenaMonitoring Framework

The main goal of the new framework is to separate the derivation of the quantities to be monitored from the histogramming of those quantities. The set of quantities to be monitored clearly requires user logic, while histogram creation and management should, as much as possible, be handled by generic code.

At the heart of the new framework is a new Athena tool, `GenericMonitoringTool`, which manages all histogram creation and filling. User code declares a set of values which can be monitored (this can include both scalar and vector quantities) and instructs an instance of the `GenericMonitoringTool` to snapshot the state of the values once they have been prepared. The `GenericMonitoringTool` is configured in the Athena Python configuration step with a list of histograms to produce using the monitored values. As this code is part of the job configuration step, the choice of histograms produced, their binning, and so on is no longer dependent on user C++ code.

This new structure addresses the structural issues with the current framework identified above. Overrides to histogram properties can easily be made at run time. User code is no longer responsible for the actual instantiation and lifecycle of the histogram objects. Changes such as a transition from ROOT 6 to ROOT 7-style histograms, or replacement of Light Weight Histograms by a ROOT 7-based solution, can be managed centrally with no impact on user code. If user code no longer handles histograms and the associated state, it becomes

Figure 3. Conceptual comparison between the current and new AthenaMonitoring frameworks. The current framework relies on user code to own histograms; the main role of the core monitoring code is to handle the interaction with the Athena THistSvc service to store the histograms in the output. In the new framework, the histograms are owned by the core monitoring code, and the user code is only responsible for extracting the values to be monitored.

much more easy to make it stateless and thread-safe. The GenericMonitoringTool itself is designed with thread-safety in mind. The initial implementation uses mutexes to protect the filling of ROOT 6 histograms from race conditions. A future ROOT 7-based implementation could use histograms based on C++ atomic types to achieve the same goal.

The GenericMonitoringTool was initially designed to meet the needs of the online trigger software community and is already in use; adoption for offline monitoring as well will reduce the number of distinct frameworks that need to be supported. The AthenaMonitoring base classes will be ported to support the new tools. A first implementation of the GenericMonitoringTool has been developed and is being tested in the development version of the Athena code. Additional capabilities need to be added in order to make it a full replacement for the existing framework (in particular the handling of time-dependence), but no serious issues are foreseen and the code is expected to be delivered in time for ATLAS integration tests in 2020 ahead of Run 3.

Several features of the current AthenaMonitoring framework were essentially never used, in particular those providing metadata for histograms. In fact the only metadata routinely used is that used to choose the method used to merge histograms from different jobs. Unused metadata fields and other unused features will be removed from the framework in Run 3.

Because users had full control over histogram objects in the current AthenaMonitoring, they have made direct use of many ROOT features. Some of these — alphanumeric bin labels, for example — are easy to implement in the GenericMonitoringTool. Some, such as the setting of ROOT graphic display options, will not be supported in the future.

4.1 The Merging Problem

Not all plots of interest can be implemented purely by accumulating statistics in a histogram. One example identified early on was a plot of the mean number of channels reporting a signal in each module of a detector; the final histogram has one entry per detector module, and the value plotted is the mean occupancy accumulated over many events. If this histogram is produced in two separate jobs, it is clear that they cannot be meaningfully merged in the final form: there is no information available on the statistical error of each point, or even which points correspond to which modules.

The impossibility of merging such histograms properly has significant practical effects. In AthenaMP jobs, histograms are produced independently in each worker process and merged at the end; histograms with merging problems have content that depends on the number of worker processes and the exact set of events assigned to each. In AthenaMT, writing thread-

safe code that correctly handles the required accumulation of data and postprocessing steps is a significant challenge

In the example given above, it *would* be possible to properly merge the output of different jobs if what was produced was a profile histogram of the occupancy versus channel number; the desired final plot is then the one-dimensional projection onto the occupancy axis. In fact, a significant category of ill-defined merging problems can be solved by increasing the dimensionality of the intermediate data. This increases the memory required but makes the workflow more flexible and the results reproducible. As the histograms will be shared across multiple execution threads, in AthenaMT it will be easier to accommodate the larger intermediate histograms required to correctly handle these cases.

5 Summary

The plans for the evolution of the AthenaMonitoring framework in the context of the overall migration to the multithreaded AthenaMT have been presented. The goal is to assist users to make their monitoring code more thread-safe by moving the details of histogram handling to dedicated core code. The implementation will also permit reconfiguration of the histograms at runtime. By leveraging the memory improvements promised by multithread-safe AthenaMT monitoring code, reproducibility of monitoring outputs can be improved by eliminating thorny problems in histogram merging.

Copyright 2018 CERN for the benefit of the ATLAS Collaboration. Reproduction of this article or parts of it is allowed as specified in the CC-BY-4.0 license.

References

- [1] ATLAS Collaboration, JINST **3**, S08003 (2008)
- [2] P. Calafiura, W. Lavrijsen, C. Leggett, M. Marino, and D. Quarrie, CHEP 2004 Conf. Proc. 456 (2005)
- [3] G. Barrand, I. Belyaev, P. Binko, M. Cattaneo, R. Chytrcek, *et al.*, Comput. Phys. Commun. **140** 45 (2001)
- [4] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, *et al.*, Comput. Phys. Commun. **182**, 1384 (2011)
- [5] S. Binet, P. Calafiura, M.K. Jha, W. Lavrijsen, C. Leggett, *et al.*, J. Phys. Conf. Ser. **368**, 012018 (2012)
- [6] I. Bird, Annual Review of Nuclear and Particle Science **61** 99 (2011)
- [7] C. Leggett, J. Baines, T. Bold, P. Calafiura, S. Farrell, P. Van Gemmeren, *et al.*, J. Phys. Conf. Ser. **898**, 042009 (2017)