Recent Improvements to the ATLAS Offline Data Quality Monitoring System

Iurii Bordulev[1], *Charles* Burton[2], *Rohin* Narayan[3], *Luka* Nedic[4], *Peter* Onyisi[2,*], and *Petronel* Postolache[5], on behalf of the ATLAS Collaboration

[1]Tomsk State University, Tomsk, Russia
[2]Department of Physics, University of Texas at Austin, Austin TX, USA
[3]Physics Department, Southern Methodist University, Dallas TX, USA
[4]Department of Physics, Oxford University, Oxford, UK
[5]Department of Physics, Alexandru Ioan Cuza University of Iasi, Iasi, Romania

Abstract. Recent changes to the ATLAS offline data quality monitoring system are described. These include multithreaded histogram filling and subsequent postprocessing, improvements in the responsiveness and resource use of the automatic check system, and changes to the user interface to improve the user experience.

1 Introduction

The offline data quality monitoring software chain [1] of the ATLAS experiment [2] at the Large Hadron Collider (LHC) played a critical role in ensuring high data taking efficiency, 95.6%, during the 2015–18 run period. The system quickly analyzes data after it is recorded, performs automatic checks, validates calibrations, presents histograms and the results of the automatic checks to shifters who perform final signoffs, archives the results for use in generating lists of good data for analysis.

During the second Long Shutdown (LS2) of the LHC, the ATLAS experiment has updated various aspects of the offline data quality monitoring software, affecting many aspects of the chain. This paper reviews key changes that have been implemented thus far.

2 Multithreaded Histogramming in Reconstruction

The ATLAS experiment is moving towards running reconstruction using the multithreaded AthenaMT framework [3] in Run 3. Offline data quality monitoring histograms are produced as part of the reconstruction step, both to access transient information that is not available in the output Analysis Object Data files, and to avoid the additional I/O that would be required if the histograms were produced in a separate process. As a result the histogramming algorithms have been migrated as part of the overall move to AthenaMT.

The overall characteristics of the new histogramming framework have been described in Ref. [4]. The core libraries are shared with the high level trigger monitoring, although additional layers are provided to provide a similar API for the offline monitoring algorithms

*e-mail: ponyisi@utexas.edu

to the one used in Run 2. The main characteristic of the new framework compared to the old system is the decoupling of user code from actual histogram filling and management. User code configures the histograms to be produced (in the Python Athena job configuration step) and provides the data to fill the histograms with (in the C++ algorithms) but does not directly interact with any histogram objects at any point. The raw histograms are handled by a common library that handles booking and rebooking, filling, interaction with the Gaudi [5] histogram service `THistSvc`, rebinning for time-dependent histograms, and so on. Fine-grained locks are used to optimize performance.

The vast majority of data quality monitoring algorithms in Athena have been migrated to the new framework. Real-world performance measurements are therefore available; it is not unusual for these to reveal issues which are distinct from those found in synthetic benchmarks. Frequent profiling with Intel VTune has been used to minimize overhead in the histogramming framework code.

The performance requirements of the offline and high level trigger applications can be in conflict, due to different typical usage patterns, and care needs to be taken to ensure that optimizations for one use case do not harm the other. One of the key features of the new framework is that a single C++ "fill" call from a user algorithm can actually trigger the filling of multiple (or zero) histograms, depending on what variables are provided in the call and what histograms have been configured. The logic that determines which histograms to fill is potentially a significant performance bottleneck, especially if invoked in inner loops; this was observed to be much more common in offline monitoring code than in trigger code (the latter tends to have many fewer histograms and simpler lookups). This performance issue is addressed in two ways. First, as the histogram configuration is not updated once the Athena job starts, a fill call with a specific set of variables will always map to a specific set of histograms to fill, which can be determined once and then cached. Second, the variables provided to the fill calls can be "vectorized" — in other words, multiple values can be provided at once, using any C++ STL collection type — and this capacity is aggressively used to move fill calls out of inner loops. For example, instead of invoking a fill call for each of the 182,500 liquid argon calorimeter cells, `std::vector`s are filled with the relevant information and only one fill call is made with the vectors as arguments. To further reduce the number of required fill calls, boolean *cutmask* variables can also be used in histogram definition to mark only certain entries of vectors as being relevant for that specific plot. This capability permits a single function call to fill histograms corresponding to different (and potentially orthogonal) selection requirements.

3 Histogram Postprocessing

It is frequently necessary to perform a processing step on a collected histogram in order to compute a desired result (for example, computing the mean of position residuals to detect problems with detector alignment). To enable multithread-safe filling of histograms, we require that histogram handling in reconstruction software consists entirely of accumulation operations. This implies that merging of histograms is a straightforward operation which merely adds bin contents, and that the order in which events are processed is irrelevant to the final histogram that is produced. It is not in general possible to merge histograms resulting from postprocessing this way. In other contexts this is part of the *map-reduce* paradigm, in which the map step is the collection of the input histograms and the reduction is the production of the postprocessed histograms. In the new histogramming paradigm only the map step is compatible with running directly in reconstruction.

The Run 2 software did include the capacity to run arbitrary code written in C++ or Python on ROOT-format [6] histogram files, but did not provide any libraries for common tasks (for example, checking for the existence of histograms, or safely overwriting existing histograms in the file). As a result the postprocessing code for different detector systems included a very large amount of repeated boilerplate code, often far exceeding the actual core logic. This also tied the postprocessing logic to a specific context (operations on static ROOT files) and prevented generalization.

The open-source `histgrinder` package [7] has been written and adopted in ATLAS as a framework to simplify user code for postprocessing histograms. It strictly separates the management of histograms (reading them from a source, writing them to a sink, pattern matching histogram names to repeatedly apply operations to sets of similar histograms) from the user logic, which is typically reduced to a few lines. Because the user logic is separated from histogram handling, `histgrinder` can be used in other contexts than offline data quality processing; a design requirement was that it could be used in the online data acquisition environment as well, in which histograms are purely memory-resident and may be asynchronously updated multiple times.

`Histgrinder` is implemented in Python. It uses YAML-based configuration files which specify input and output histogram name patterns, the name of a Python function to carry out the actual operation, and any additional function parameters. The core of `histgrinder` is agnostic regarding the actual histogram technology used and treats the histograms as opaque Python objects with associated names; input and output plugins handle the specifics of e.g. interacting with a ROOT file or the ATLAS Online Histogramming Service. (The user-provided code must interpret the histograms and is therefore not library-independent.) A regular expression-based pattern matching mechanism is provided to enable workflows such as repeating the same postprocessing operations for histograms corresponding to different subdetectors, slices in η and ϕ, different reconstructed object quality and trigger selections, and so forth. The `histgrinder` engine operates in “streaming” mode: it does not require all histograms to be available at once, but can handle asynchronous arrival of histograms (as occurs in the online environment), emitting new histograms as pattern matches are fulfilled.

4 Reference Histogram Updating

The histograms produced in reconstruction and any subsequent postprocessing are then analyzed by the offline Data Quality Monitoring Framework (DQMF), which is a library that runs check algorithms on the histograms according to a provided configuration. The configurations are strongly versioned and standalone: the “binary” configurations are stored as custom objects in ROOT files, along with all necessary reference histograms for the check algorithms and for user display, and the results of the DQMF application can be reproduced solely from the binary configuration and the input file of histograms. The binary files are compiled (manually) according to the specifications in text files which reference external ROOT files for the references; the text files are versioned via `git`. The specific version of binary configuration to use for a specific run is determined by the same ATLAS Metadata Interface (AMI) database [8] that configures reconstruction tasks.

The Run 2 binary configuration production and versioning system is designed to be robust and reproducible; the downside is that they are not flexible and cannot be quickly changed according to user needs. Updating the configurations, in particular changing the reference histograms, often took multiple days in Run 2. To address the reference histogram issue in particular, an update to the offline binary DQMF configuration production and versioning has been implemented during LS2.

Although the text DQMF configurations are technically able to use a different reference file for every histogram, in practice there are a few specific reference files used by large sets of histograms: one for trigger monitoring histograms and one for all others, with variations depending on trigger stream and type of data (*pp* collisions, cosmic ray data, heavy ion collisions). These reference files can now be specified using the ATLAS conditions database, implemented using the COOL technology [9]. This allows the specification of a particular set of reference files to be used for a specific run in a permanently-versioned manner. One attractive feature of the ATLAS COOL interface for this usage is the ability to prohibit retroactive changes: the stored data for a specific run can be declared to be modifiable until the run actually starts or until the run starts reconstruction after the 48-hour calibration window, after which the data are locked. This structure means that specifying the text configuration, the database information, and the run number is enough to reproducibly create the binary configuration. Additional versions can be added as needed, for example to update references for a year-end reprocessing of data.

At the start of every run and at the start of the final processing of the data, the database is checked for updates; if an update has occurred a new binary configuration file is generated and used. A web interface has been prepared to perform the database updates. ROOT files to be used as references must first be installed in a dedicated write-once directory. The COOL versioning mechanism means that the requested references can be changed as necessary for runs in the future, but not for data which have begun to be reconstructed.

5 Data Quality Result Storage

The results of the offline DQMF are permanently archived in ROOT files, which are used as the basis for all user display and database query applications. Many of these files are produced per run: a separate file is generated for each 20-minute subrun interval for histograms configured to be monitored with that granularity, and each monitored trigger stream and Tier-0 reconstruction pass generates a new set of files. This leads to two areas of concern: the use of a very large number of filesystem inodes due to the large number of files (over 2 million) and the overall storage requirements (around 20 TB for Run 1+2 data). In order to avoid hitting resource limits in Run 3, an optimization campaign has been performed.

The number of files has been reduced by the simple procedure of placing the ROOT files into ZIP archives [10]. As the ROOT files are already compressed, no compression is used when creating these archives, which purely serve to bundle multiple ROOT files together into a single file which still provides quick random access. The native ROOT support for reading files in ZIP archives makes this simple to implement. This reduces the number of required inodes by better than an order of magnitude.

The amount of space required is being optimized by revisiting the file format for the result storage. The current file format places a heavy emphasis on random access to individual histogram results, using a tree structure based on ROOT `TDirectory` objects which in many cases contain a single `TObjString`. In fact the ROOT `TDirectory` has significant on-disk overhead due to preallocated (but, in our case, unused) directory entries and is not compressed, leading to disk use of the same order as that taken by the (compressed) histograms themselves.

This is being addressed by an update to the file format which replaces most of the `TDirectory`/`TObjString` pairs by `TObjString` objects containing JSON data. This is able to reduce the file size by 52% while not significantly altering access times, even considering the parsing of the JSON data.

6 User Presentation

The results of the offline DQMF are presented to users in a number of ways, but the primary interface is the Web Display, a Python-based web application with a REST API. Historically

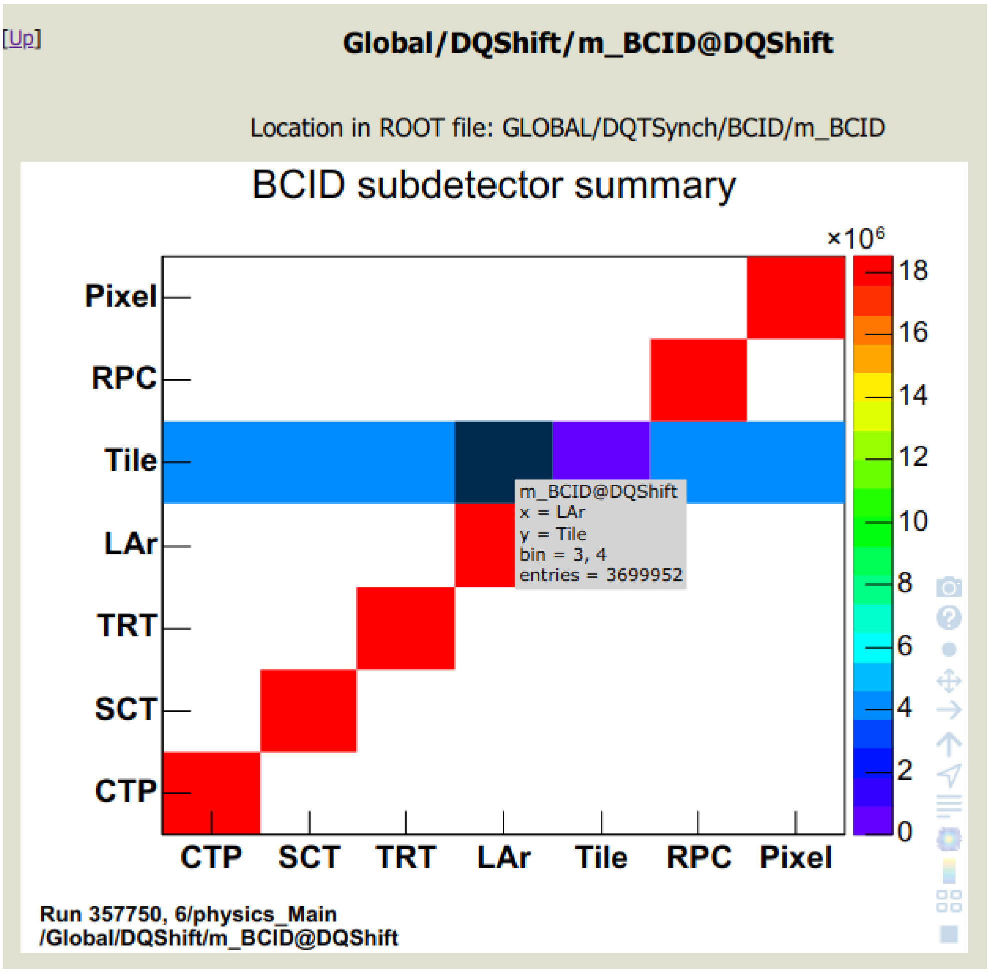

Figure 1. Example of a JSROOT-rendered histogram in the Web Display application. The interactive JavaScript code provides tooltips that show information about the bin that the mouse cursor is hovering over. Additional functionality to change the style of the plot (palette, logarithmic vs. linear axes, and so on) are available via the toolbar on the right.

the HTML and PNG for the histograms were generated on the server, and the browser merely displayed the page. This meant that all histogram rendering attributes had to be set on the server as part of the DQMF configuration.

To enable more flexibility in the web display, we have moved to rendering the histograms in the browser using the JSROOT package by default [11]. The initial rendering of the histogram still happens on the server-side to maintain compatibility with the existing configurations; the resulting `TCanvas` object is serialized to JSON and transmitted to the browser, where it is turned into a dynamic SVG image by JSROOT. The (compressed) JSON representation is typically smaller than than the corresponding PNG by a factor of 2-5. The JSROOT rendering speed was considered insufficient for parts of the web display application that show histogram thumbnails; for these pages the PNG rendering path is still used.

Users also desired the ability to change the displayed reference histograms to correspond to arbitrary user-selected runs. This has been implemented with a REST-based URL endpoint that can change the reference to be the result of an arbitrary run, reconstruction pass, or trigger stream. Because the DQMF results are not in general valid if the reference is changed, only the histogram and the reference are shown. As above the default is to use JSROOT to render the plots as interactive SVG.

7 Summary

Improvements to the ATLAS offline data quality monitoring system during the LHC Long Shutdown 2 are motivated by changes in the ATLAS offline software framework, the need for additional flexibility, resource constraints, and the desire to enhance the user experience for data quality shifters. These changes affect many components of the system: histogram production and processing in the reconstruction software, the handling of reference plots, the storage of the results of automatic checks, and display of the histograms and results to shifters.

References

[1] ATLAS Collaboration, JINST **15**, P04003 (2020), `1911.04632`
[2] ATLAS Collaboration, JINST **3**, S08003 (2008)
[3] C. Leggett et al., J. Phys. Conf. Ser. **898**, 042009 (2017)
[4] T. Bold, W. Lampl, R. Narayan, P. Onyisi, P. Sarna, EPJ Web Conf. **214**, 02041 (2019)
[5] G. Barrand et al., Comput. Phys. Commun. **140**, 45 (2001)
[6] R. Brun, F. Rademakers, Nucl. Instrum. Meth. A **389**, 81 (1997)
[7] *Histgrinder project*, https://github.com/ponyisi/histogram_postprocessing/, accessed 2021-02-27
[8] J. Fulachier, J. Odier, F. Lambert, J. Phys. Conf. Ser. **898**, 062001 (2017)
[9] M. Verducci, J. Phys. Conf. Ser. **119**, 042031 (2008)
[10] *PKZIP application note*, https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT, accessed 2021-02-27
[11] *JSROOT project*, https://root.cern.ch/js/, accessed 2021-02-27