Activity Report
Internship at CERN

2008/2009

Ricardo Abreu
ricardo.abreu@cern.ch

September 29, 2009

Supervised by Joannes Andreas Bogaerts.

Thanks to Tomasz Bold, Werner Wiedenmann.
Special Thanks to André dos Anjos.

Abstract

During this past year at CERN I have been working in the ATLAS experiment, concretelly in the HLT and TDAQ projects. I am currently developing and maintaining two packages, each one belonging to one of these systems. The first one – HLTTestApps –is the last means for testing HLT algorithms before they are put into operation in the ATLAS online systems. It provides the testing enviroment that stands closest to the real system. The second one – histmon – has the responsibility of publishing histograms produced by the TDAQ applications. Histograms are one of the two available means for monitoring online operations and histmon is necessary for them to become available to the users.

This document summarizes my work at CERN during this period, drawing from the knowledge I have been gaining to concisely expose some of its grounding concepts, and pointing out my main accomplishments.

1 Introduction

ATLAS is one of the four experiments of CERN's Large Hadron Collider [1]. As one among the thousands of people involved in it, I have been actively contributing to the experiment, since November 2008, within the scope of the existing protocol between CERN and the portuguese agency AdI.

Since my scientific background is computer science, my work in CERN is bound to the computational aspects the experiment relies on, focusing particularly on a couple of software packages that integrate the Data Aquisition (DAQ) and High Level Trigger (HLT) systems [2]. These two packages – named HLTTestApps and Histmon – can be seen as bounded units of software, conceived to achive specific goals and provide certain features. Developing and maintaining them is what I have been spending most of this last year's time on.

Nevertheless, the nature of their common purpose requires them to maintain interdependent relations with other software packages that are part of the HLT/DAQ system (much like organs in a body), in a way that impacts their configuration, implementation, and running processes and environments, as required by the operation of the system as a whole. This is reflected in certain traits of elaboration that are not shared by isolated software projects. Together with the dimension of the DAQ/HLT project, with the advanced character of the field it leans on (particle physics), and with a small but present degree of inate subjectiveness on the human communication around it, this fact accounts for a long learning curve for newcomers. For this reason, a large chunk of my time has been spent on trying to grasp all the concepts and to untangle their relations, to build a solid understanding that is constantly growing and being sharpened.

I believe this also explains why the wish for detailing my recent work in this document conflicts, to a certain extent, with the need for it to be succinct. Therefore, the remainder of this report exposes only an abstract overview of the most important aspects of my activity at CERN, along with a broad clarification of the grounding concepts behind it.

1.1 Grounding Concepts

With the ATLAS detector at its core, the ATLAS experiment will search for new discoveries from the results of collisions of protons of very high energy, with the purpose of testing and increasing the notions that primarily underly our current understanding of the Universe. The detector is actually composed by a set of subdetectors that focus on diferent aspects of the outcome of the proton collisions, so that the domain of the potential studies and results is broadened to encompass what is usually called "general particle physics", in contrast with LHC experiments that concentrate on specific issues. Added to the fact that collisions happen at an incredibly hight rate, this means that the data gathered per unit of time amounts to incredible levels. Neither storing it all nor having it all analysed by physicists are feasible ways of tackling the matter. Instead, irrelevant data has to be filtered from the results that matter, according to a set of rules that is determined by a so called *physics menu* and enforced by the operation of specific algorithms. The DAQ/HLT system provides an appropriate environment for the execution of algorithms that respect the established interfaces, ensuring a carefully designed data flow from the detectors to the offline storage facilities, where it waits for further and deeper processing.

The data that is obtained from the detectors is conceptually organized in *events*. At this initial phase, the term "event" describes the set of data that was collected during one of the *bunch-crossings* (plus any delayed data from previous bunch-crossings). As it travels through the different steps of the dataflow path, an event is either discarded, if concluded to be irrelevant, or kept and enlarged with more data that is produced in its analysis, if considered potentially interesting.

The $14TeV$ proton-proton bunch-crossings happen once every $25ns$, that is, with a frequency of $40MHz$ [3] and, out of the corresponding events, only about 200 per second can be recorded for offline processing. The first step of this drastic frequency reduction is performed by the custom hardware that constitutes the *First Level Trigger* (*L1*). Its output reaches the DAQ/HLT with a frequency of about $100KHz$. The subsequent reduction is achieved by the two levels that compose the HLT: the *Second Level Trigger* (*L2*) and the *Event Filter* (*EF*)[1]. L2 delivers a maximum $3KHz$ event output rate, while the EF cuts it down to the required $200Hz$. Considering the average event size (about $1.6Mbytes$), the network configuration (based on Gigabit Ethernet), and the number of CPUs currently dedicated to L2 and EF tasks (about 500 and 1800, respectively), each event is processed by L2 in an average of $40ms$, which contrasts with the time the EF is allowed to spend in each event – about $1s$ on average. The main rationale behind this design is to be able to discard obviously irrelevant events as soon as possible, so that more time is left for deciding what to do with data that demands more attention.

Both L2 and EF subsystems depend on more than just the applications that do event analysis. As parts of a larger system, they need to interact and interface with it in a fashion that doesn't compromise its strict real-time requirements. Therefore, they rely on applications that are responsible for distinct but equally important tasks, such as assigning events to processing tasks, accepting and providing for demands of further analysis required data, and packing different tracks of data in "ready-to-ship" events. Applications that fullfil these other requirements include, but are not limited to, the *Read Out Buffers* (ROB), the *Region of Interest Builder* (*RoIB*), the *Level 2 Supervisor* (L2SV), and the *Data Flow Manager* (*DFM*). They are, in some cases, grouped in subnetworks and subsystems, such as the *Read Out System* (*ROS*), the *Event Filter Network* (*EFN*), or the *Event Builder* (*EB*). These are the main components that integrate the HLT and the DAQ systems, following a division that owes more to project conception and other historical reasons than to a clear taxonomical division. An overview of the *ATLAS Trigger*, with a focus on the DAQ/HLT systems, is depicted in Figure 1.

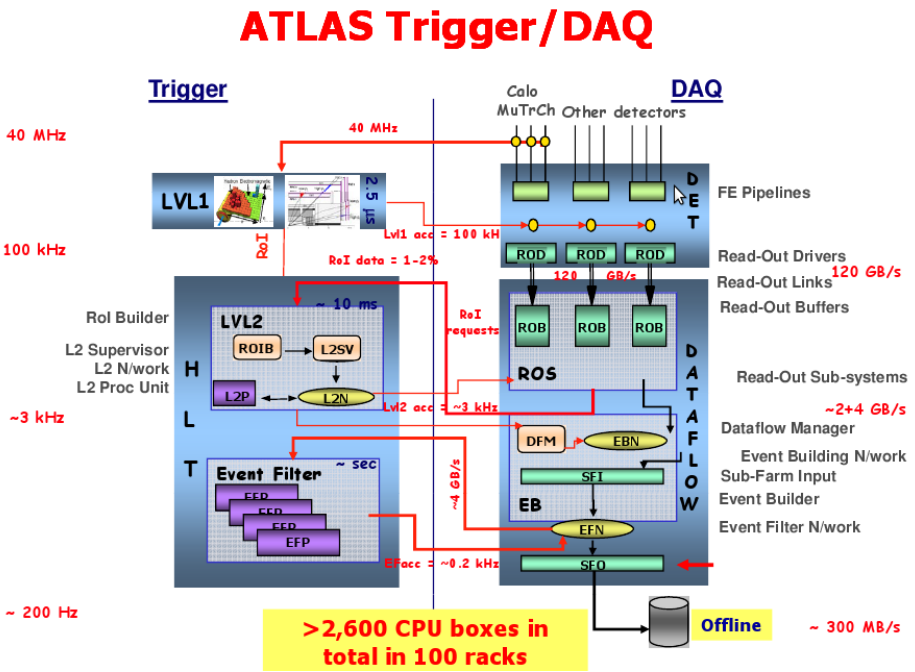

Figure 1: Overview of the Atlas Trigger and DAQ systems

[1]The names might be misleading since all the three levels perform event filtering, in their own way. In such a large and long lasting experiment, names do not always precisely describe the concepts they intend to identify, but they are justified for historical origins

2 HLTTestApps

Besides reading and otherwise learning about ATLAS and DAQ/HLT, the first task I was assigned after comming to CERN was to maintain and develop the HLTTestApps package. I thus became a co-developer of this package, along with its previous developer (and current co-developer) André Anjos, who has been tremendously helpful in orienting me on several matters ever since.

HLTTestApps is included in the HLT project and, as such, sits on top of the *Trigger/DAQ* (*TDAQ*) and *AtlasOffline* projects, participating in the effort of integrating them by providing general test facilities for both the AtlasOffline's algorithms and TDAQ's applications (see Figure 2).

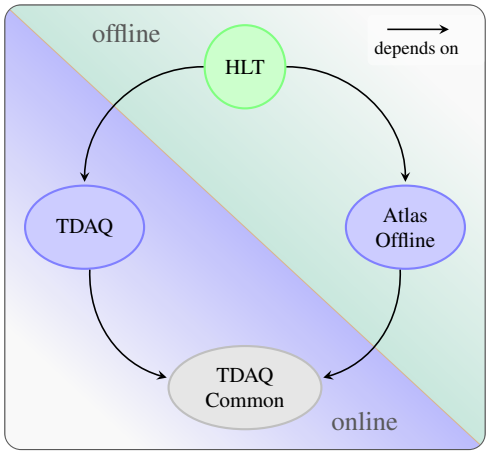

Figure 2: Dependency relation between HLT, TDAQ, and AtlasOffline

The HLTTestApps package currently provides two tools that are broadly used in the ATLAS collaboration: *athenaMT* and *athenaPT*. AthenaMT emulates the operation of L2 while athenaPT emulates the operation of EF. Both commands accept several options that allow the user to configure many aspects of their operation. According to the user's choices, they can interface with the processing applications alone or create a small *localhost* partition[2], so that they can extend their functionality to include some more of the features provided by the TDAQ software.

The command line interface is similar to the well known offline *athena* application and, in many cases, uses the same syntax. Both applications are written in *Python* and, to a large extend, share a common code basis. This code basis is again mostly written in Python but it also includes some *C++* code that acts as the "glue" between the tools and the software they sit on (whose interfaces are provided in C++). The C++ code is there essencially to bind C++ interfaces to python.

2.1 AthenaMT/PT

AthenaMT works as an interface to execute the Level-2 HLT online selection code as an offline application. It provides the same environment for algorithms as the real *L2 Processing Unit* (*L2PU*) application. For algorithm testing, it is equivalent to a data flow system composed of a single L2 node. It does however not require the user to setup and run a separate supervisor to control the L2PU, nor a ROS application to provide it with data. It doesn't require the creation of a data flow configuration for a single node system either. At startup, athenaMT scans the provided byte stream data file for event data and stores the ROB

[2]In simple terms, the word partition is used to refer to a particular running configuration of the applications that compose the ATLAS Trigger and DAQ systems.

fragments from a given event in memory. ROB fragments from Level-1 with RoI information are stored for each event in a separate queue, like in the real L2PU application. For event processing, Level-1 ROB fragments from an event are retrieved from the queue and given to the algorithm framework. Depending on the decoded RoI information, the algorithms will then retrieve detector data from memory via an interface that emulates the real L2PU's data collector. After processing an event, the Level-2 decision and result, which summarize the HLT algorithm processing, can be saved to a file, in byte stream format, together with the full event fragment, for further investigation. Such output files can be used as input for Event Filter tests where Event Filter algorithms use the Level-2 result as seed to start event reconstruction. AthenaMT can also be run in monitoring mode, connecting to a running partition, subscribing to streams of events, and consuming those that match selected criteria.

AthenaPT is the Event Filter counter part of athenaMT and provides a framework to test and validate Event Filter trigger algorithms. AthenaPT works similarly to athenaMT, providing the same environment for algorithms as the real *EF Procesing Task* (*EFPT*) dataflow application. Like in athenaMT, a separate supervisor is not required. AthenaPT emulates the input from a single Event Filter node by using a byte stream data file, which corresponds to the output of athenaMT. When running, athenaPT retrieves the event data from memory via an interface that emulates the real PT's data collector. After processing an event, the Event Filter decision and result can be saved into a file in byte stream format, together with the full event fragment, for further investigation. As athenaMT, athenaPT can also be run in monitoring mode.

During my stay in CERN so far, I made several changes and updates to athenaMT and athenaPT, with different importance and impact. The most important ones were:

- Added the possibility of selecting only certain events or streams, based on low-level characteristics (e.g. *streamtype*, *streamnames*, *streamlogic*, *l1bits*, *l1logic*, and *l1origin*).
- Implemented functionality for receiving and forwarding *user commands* to the HLT. The *hltUserCommands*, along with their arguments, can be provided directly to an athenaMT/PT process between state transitions. For executing these commands at other moments, a separate client that connects to a multiple connection server can be used.
- Added support for Google's TCMalloc library[3].
- Added fault tolerance for handling events with invalid ROBFragments.
- Consistently implemented automated unit tests for several features.
- Implemented the creation of event streams from several files that can have different run numbers

My involvement in the package HLTTestApps also included the partial development of another tool, called athenaXT. This tool is intended for having the streams of events processed by L2 and EF in a single run, in multiple simulated L2 and EF nodes, while providing the same features as athenaMT and athenaPT. For the multiple parallel processing, athenaXT must rely on a multi-process architecture. Due to subsequent progress on the *TDAQ Software Upgrade* project, which comprises the concept of *xpu* nodes that can play the roles of the current L2 and EF nodes, and due to other more pressing priorities, the development of this tool is currently halted and unfinished. The most important developments I made so far are summarized in the following list:

[3] http://goog-perftools.sourceforge.net/doc/tcmalloc.html

- Implemented a general worker process whose members can be accessed from a handle in the parent process.
- Included support for logging and outputting to custom ERS streams[4], installing a backend that transforms the worker processes in ERS clients. This prevents incorrect mixing of output from different processes.
- Created and configured the *HLTTestApps_ERS* python sub-package, to isolate HLTTestApps' ERS features.

Smaller (though often equally important) changes that are related with the maintenance of HLTTestApps are ommited from this report, since they require a deep understanding of the package and could not be exposed without intricate code references.

3 Histmon

Contrary to HLTTestApps, which is part of the HLT project, *histmon* is a package that belongs to TDAQ. Because it is required for the publication of histograms produced by other applications in the Trigger and DAQ systems, this package is essencial for the ATLAS experiment to be fully productive. Along with the so called *counters* – discrete quantities that measure some aspect of an application's operation – histograms are the main vehicle of online monitoring data for ATLAS.

Rather than defining an application in itself, histmon provides public interfaces that can be used to plug its functionality in the target applications, along with the corresponding applications. Histmon consists of mainly two logical parts: a register for histograms where applications can put the histograms they produce and a mechanism for publishing these histograms, so that, in the end, they can reach the *Online Histogramming* (*OH*) infrastructure and be examined in the ATLAS control room.

TDAQ applications, such as the L2SV or the EF, register and fill their histograms through an interface that is provided by the *hltinterface* package and implemented in histmon. Histmon then publishes the histograms in the closest *IS*(*Information Service*) servers, with periods specified for different histogram types in the partition configuration. Akin histograms, published by different instances of the same applications, are then summed by the *Gatherer* application and forwarded to the next IS server in the hierarchy. On the top level, summed up histograms – that result from summing hundreds of thousands of individual histograms – are put in the same IS server.

I recently (at about June 2009) became the new developer/maintainer of this package. Its previous developer – Tomasz Bold – is currently still contributing to its maintenence, although his intervention is being phased out, as I become more familiar with the package. All my development in the package was oriented by him and by André Anjos. Succintly enumerated, my contributions so far were:

i) Limit the interface that is shown to clients and hide the implementation outside of the public folder, to maximize what can be changed with patches.

ii) Review how the schema of priorities affects the scheduling of histograms to be sent out, introducing parallelization, to make sure that a reasonable balance is achieved and that no histograms are withhold for too long.

iii) Simplify the implementation and prune what is obsolete.

[4] http://atlas-tdaq-monitoring.web.cern.ch/atlas-tdaq-monitoring/ERS/doxygen/tdaq-common-01-07-01/html/index.html

iv) Review and complete the documentation (ongoing).

v) Disallow interactive binning commands – only initial binning commands are allowed.

vi) Eliminate the throwing of exceptions (which would crash the main application) when the configuration cannot be loaded for some reason. Still, prevent subsequent calls from AppControl from having any effect and the threads from running, in this case.

From these changes, the most important were i), ii), and v). With i), the patching procedure is now more flexible and allows us to provide patches with corrections that were previously restricted. In fact, the code shipped in patches cannot change the interfaces that are available in the package's public folder, since these are *linked* by external packages that are not compiled after the patches are applied. Therefore, patch changes must be limited to the code that lies in private folders. To achieve this, "public" classes were made shallow, in the sense that they have only one private member that holds their implementation. Objects of these classes first forward every operation to the implementation object they hold. After that, any results yielded by the implementation are forwarded back to the caller. By placing the implementation exclusively in the private folders, the range of code that can be changed through patching was highly increased and no features were perturbed.

Histmon can receive commands that are forwarded from the user interface that displays the histograms, through the partition network, until they reach the target application. These commands can be placed in two categories: *Initial Commands* and *Interactive Commands*. Initial commands are those that are received before the run has started. Interactive commands are received after the start of the run. The commands are processed by histmon to achieve the desired result, either locally or by relying on features of the underlying libraries. One particular class of commands – the *Binning Commands* – somehow affects the binning of the histograms. The binning is changed when the methods of the underlying *Root* library[5] for that effect are called. Unfortunately, these methods are not thread-safe and, since, after the start of the run, histmon has its tasks performed by threads, Binning Commands cannot be issued as Interactive Commands. Otherwise, the penalty would be undefined behavior, which must be avoided at all costs. Thus, histmon now issues an error if Interactive Binning Commands are received. The change v) refers to the fact that only Initial Binning Commands are now allowed.

Before I started working on this package, histmon used only one additional thread to perform its publishing duties. Publishing histograms that required different publication periods was something that was done sequencially, by a single thread. In principle, this approach has no problems, as long as publishing one type of histograms does not take longer than any of the frequencies required by other histogram types. But, when the number of histograms that are published by a single application reaches the levels that are currently seen in the TDAQ system, this constraint cannot be guaranteed. Hence, some months ago, histograms' publication periods were often beeing disrespected.

At an initial approach, and because no bigger changes were allowed in patches at the time, Tomasz Bold devised a quickfix that solved this problem in a sequential fashion, at the expense of performance. The main idea of this preliminary solution was to have a queue of histograms awaiting publishing, where they were ordered according to their priority. Unfortunately, for this queue to be always ordered, any histogram that was inserted into it had to be inserted in the right position, and this process is very time consuming. Tomasz was perfectly aware of this and intended to implement a parallel solution that would solve the same problem while maintaining high efficiency, when he got the chance, that is, when the new release of TDAQ happened. It happened that the next TDAQ release was scheduled for a time when I was already familiar enough with histmon to be able to implement this solution myself.

[5] http://root.cern.ch/drupal/

With the new approach, which is now in successful effect, we have one thread per histogram type, so that each thread has to fulfill only one publication period requirement. Most of histmon's tasks are delegated in these threads by another thread that manages them. This global thread maintains an index of the other threads and routes each opperation to the right one.

In order not to have unnecessary idle threads (which would still mean an undesirable overhead) in every application, a new field was included in the configuration schema object that describes the different histograms types. This field indicates which applications are targeted by a histogram type, which allows histmon to select what types to consider and create threads for, since external "reflection" methods are available to tell it in which application it is currently running. The tool that is used to automatically create partition configurations was also modified, by its developer André Anjos, in order to take these new changes into account.

To validate the new solution, the CPU usage of athenaMT was measured when using the old and the new versions of histmon. For this, I created a python script that accesses values, produced by the UNIX opperating system, that indicate how many clock ticks are used by any running process. This script is able to process these values and produce a *comma-separated value* file where the percentage of CPU used by the measured process is indexed by time. Running this script with a sampling interval of $1s$ (like the *top* UNIX program[6] does by default), and using the output file to produce a plot with *gnuplot*[7], the plot of Figure 3 was obtained.

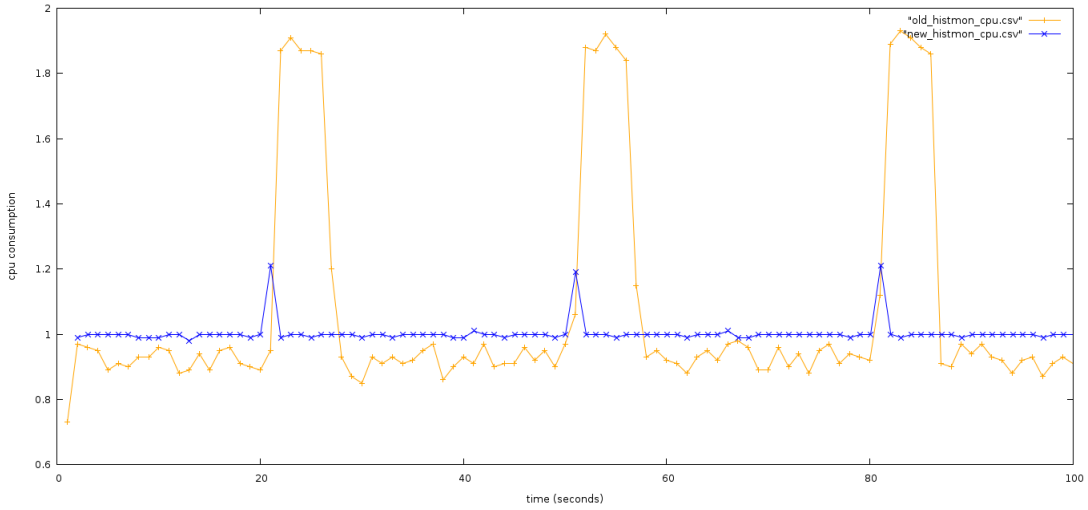

Figure 3: Comparison of athenaMT's CPU usage when using old and new versions of the histmon library.

The plot marks CPU consumption in the Oy axis, as the unitless quantity given by the ratio $\frac{U}{E}$, where U is the number of clock ticks used by athenaMT and E is the number of click ticks elapsed. The Ox axis measures the time, in seconds. The analisys of this plot yields three main conclusions.

What are the peaks? The peaks that can be seen both in the yellow and blue graphs start at approximately the same time and the timespan between the beginning of any two consecutive peaks is the same in both versions. Moreover, the time that elapses between two consecutive peaks is the same as the publishing period that had been defined for publishing the histograms of the main histogram type. Because of this, it is safe to assume that the peaks correspond to the processing that is done for publishing histograms. The other

[6] http://www.unixtop.org/
[7] http://www.gnuplot.info/

histogram type that was used for these runs of athenaMT included only a neglectible amount of histograms, which explains why only one type of peak is observed in each plot.

What are the fluctuations? In the ordered send queue version of the plot (the yellow one), some fluctuations can be seen in between the peaks, while almost none are present in the multi-threaded version (the blue one). These fluctuations drive the processing under 100% CPU usage. The changes in histmon do not justify this result, since histmon is mostly idle when not publishing (no commands were sent to it). Still, in these periods, athenaMT should have pleanty of work to do in event processing. Since, besides histmon's threads and the user command server thread, which is also idle, only the main athenaMT thread is running, in the absence of external strangling factors, the processing should always be very close to 100% – one full processor core. Indeed, external factors are the exact justification for these fluctuations. In my opinion, the most probable influences were the activity of other users in the same machine and the absence of the events in the filesystem cache.

What was the improvement? The first thing that is noticeable when observing the graphs is probably the fact that some peaks of processing remain in the latest version. It might seem that this is a negative result but, in my opinion, this would be an incorrect conclusion. In fact, the peaks correspond precisely to the moments when there is activity of any thread besides the main one – namely, that of histmon's threads. As athenaMT was executed in a multi-core machine, it is only natural for more cores to be used when histmon is publishing histograms. This contributes to a higher performance and not the contrary. Still, the blue peaks rise little and decrease very quickly, when compared to the yellow ones. This means that, though histmon had to complete the same amount of work in both cases, it was completed much faster with the new version. Considering that the number of clock ticks used by the histogram publishing tasks is given by the extra CPU used times the time spent, histmon's performance was improved as much as the area just bellow the yellow peak is larger than the area just bellow the blue peak.

4 Conclusion and Future work

The time I spent at CERN so far has been very productive, in more than one aspect. It allowed me to learn a lot about the very interesting ATLAS experiment, generally, and about the HLT and TDAQ systems, more specifically. I have been benificting from the scientific environment I work in every day, acquiring general and specialized knowledge, not only related to computer science but also to human resources and project management. I profit from being able to interact with people I can learn a lot from and I am glad to be able to work with them.

Besides being generally involved in the ATLAS collaboration and in the HLT and TDAQ groups, I have been contributing mainly by developing and maintaining two packages. One of them – HLTTestApps – allows other people to test their HLT algorithms with real TDAQ applications, in an effort for integrating both worlds into an operable seamless system. The other one – histmon – is a part of the TDAQ system itself and is essencial for the publication of histograms, which constitute one of the two main ways for monitoring the most diverse aspects of ATLAS's operation.

From the feedback I get, I believe my work has been welcomed as useful and successful. I look forward to expand the reach of both my knowledge and my experience in one more year of work at CERN.

References

[1] G. Aad et al. The ATLAS Experiment at the CERN Large Hadron Collider. *J. Instrum.*, 3:S08003, 2008.

[2] F. J Wickens et al. The ATLAS HLT, DAQ & DCS Technical Design Report. Technical report, CERN, 2003.

[3] André dos Anjos et al. The DAQ/HLT system of the ATLAS experiment. In *Proceedings of Science PoS(ACAT08)*, 2008.