PROCEEDINGS OF SCIENCE

ATLAS Handling Problematic Events in Quasi Real-Time

Hegoi Garitaonandia*
NIKHEF
E-mail: Hegoi.Garitaonandia@cern.ch

Sander Klous
NIKHEF
E-mail: Sander.Klous@cern.ch

Brian Petersen
CERN
E-mail: Brian.Petersen@cern.ch

Anna Sfyrla
UIUC
E-mail: Anna.Sfyrla@cern.ch

In this paper we present a new fail-over framework for the ATLAS online system. The main goal of this system is to identify problems as soon as possible and reduce the turn-around time for fixing these problems. Design, implementation, performance and results are discussed in detail.

XII Advanced Computing and Analysis Techniques in Physics Research
November 3-7 2008
Erice, Italy

*Speaker.

PoS(ACAT08)031

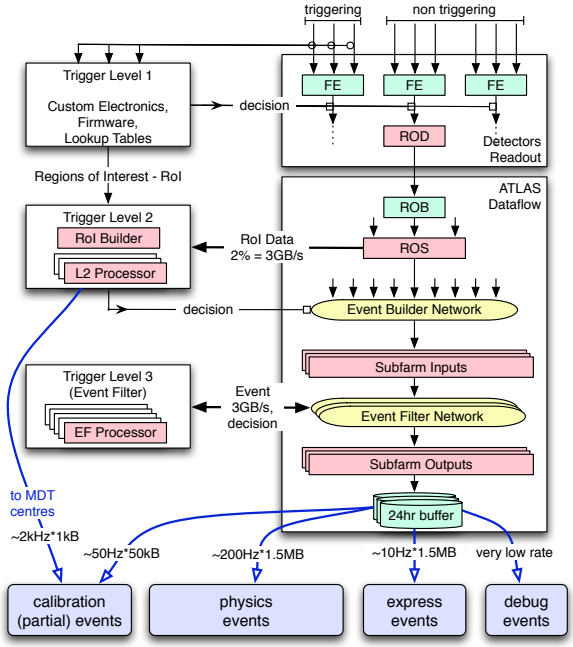

Figure 1: Architecture of the ATLAS Data Acquisition System. Data flows from detector (top) to mass storage (bottom). Left side represents processing and selection, while right side represents data movement.

1. Introduction

The Trigger and Data Acquisition system (TDAQ) of the ATLAS experiment at the CERN Large Hadron Collider is based on a multi-level selection process and a hierarchical acquisition tree. The system, consisting of a combination of custom electronics and commercial products from the computing and telecommunication industry, is required to provide an online rejection power of 10^6 and a total throughput in the range of Terabit/sec[1].

The TDAQ system is affected by many types of runtime errors, such us power cuts, networking timeouts, operating system failures, crashes, and algorithm errors (possibly due to the processing of new physics). Besides, in a system of such dimension [1] the probability of an element failing is high. Redundancy and fault tolerance were therefore key design points of the ATLAS TDAQ. As a result of this approach, not all errors force data taking to be stopped. Instead, when a minor error occurs, the data associated to that error is marked. The subsequent elements in the TDAQ chain then route that event to a special collection of events, without performing any further processing. This disjunctive group composed by all the problematic events is called *the debug stream*.

Even though the problematic events are expected to represent only a small fraction of the overall amount during normal operation, it is necessary to record and understand them. Discarding them would imply loosing all the information that is necessary to debug the corresponding failure. It would also introduce the risk of creating a bias in certain physics channels. Keeping and analyzing the debug stream is of major importance during first beam commissioning and early data taking, as unexpected issues are likely to occur. Treating the debug stream as soon as possible allows to identify problems and reduce the turn-around time for fixing them.

Figure 1 introduces a more detailed view of the TDAQ system. Data is moved from the detector front-end electronics, in the upper part of the picture, to storage, in the bottom of the picture. The upper left part shows the three decision levels of the hierarchical event selection. The first decision level (L1) is implemented in hardware, while the second (L2) and third (L3, usually called Event Filter or just EF) run in dedicated computer farms. The right part shows the elements of the TDAQ system that are necessary to transport the data to storage. They feed the selection elements shown on the left with data, but they do not participate in the event selection process itself.

When the events arrive to storage, they are divided over different locations, according to the streaming information. In normal operation the system is configured with the following stream definitions: *physics* (events relevant for physic studies), *debug* (events that experienced an error), *calibration* (events that are relevant for detector calibration), and *express* (a copy of a small fraction of the physics events for fast offline data quality monitoring)

Inside the debug stream type, at least three different sub-streams are usually configured: *debug.HLTError*, *debug.L2Error*, and *debug.EFError*. The first of these sub-streams corresponds with events that could not be correctly processed by the selection algorithms (*e.g.*: software exception, readout problems, etc). The second and third sub-streams are associated to online errors, like timeouts or crashes. The index L2/EF indicates the selection trigger level where the error occurred[1].

By re-running the selection algorithms with no time constraints in the *debug.[L2/EF]Error* one can try to recover a significant fraction of the events. The recovered events can be assigned a correct physics tag.

To understand and recover the events in *debug.HLTError* we need a longer turn around cycle, as they usually imply software fixes. First the different type of errors must be classified. Then the bug fixing process can be triggered (report, assignation, fix by expert, validation, installation, test).

In order to handle the debug stream we developed a software package called TriggerOffline. In section 2 we describe its architecture. In section 3 we present some of the results obtained with cosmic runs.

2. Quasi Real-Time Handling of Problematic Events

In the data chain, the TriggerOffline system is located just after storage of figure 1. It performs two basic operations on the *debug* stream as it comes out of the TDAQ system: fine grained error classification and rerunning of selection algorithms. Most data exchange between TriggerOffline and other ATLAS subsystems is performed through the Castor storage system at CERN[2]. A file catalog[2] is used for synchronization purposes, exchanging meta-data and bookkeeping.

The runtime procedure is described in figure 2. The manager queries the file catalog to check whether the TDAQ system created more debug stream files. If new files became available, the manager schedules several jobs in a batch queue, with links to the corresponding files. Parallelizing this process is trivial since the events are independent from each other. The jobs copy the input data from storage, process it, and then copy the output back to storage. The manager is in charge

[1] Note that there is no *debug.L1Error* because the first online selection level is implemented in hardware.

[2] The file catalog is an application-specific solution implemented directly on top of Oracle. It has nothing to do with the Grid-like file catalogs.

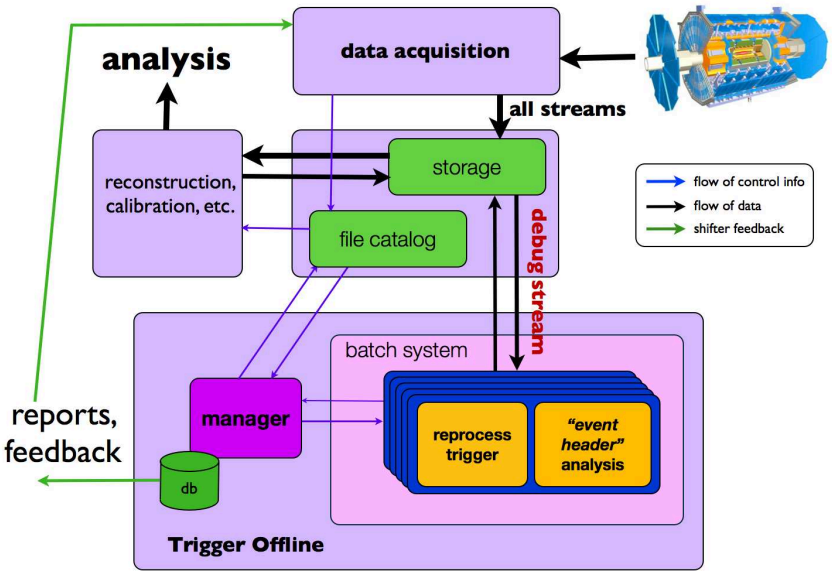

Figure 2: Overview of TriggerOffline in its context.

of keeping track of the status of the jobs, re-submitting failures, etc. In order to present the results of this automatic process to the shifter, the manager creates a web report for each run. The quality of the data is evaluated and problems are diagnosed with this report.

The first task is the *event header analysis*. Its objective is to analyze the contents of the debug stream and provide fine grained error classification. For this, the *event header analysis* first extracts information that has been recorded in the event header during data taking, such as the active algorithms and their processing timing. As a second step, abundance and several correlations are studied. When this procedure is applied to the *debug.[L2/EF]Error* streams, it gives information about aspects that could help identifying problems, *e.g.:* periods of run with bursts of errors. When applied to the *debug.HLTError* stream it gives information about algorithm problems.

The second task that the job performs is reprocessing the events with the L2 and EF selection algorithms. The idea behind this is that the errors that pushed the event into the *debug.[L2/EF]Error*, could be transient or cumulative, like timeouts or crashes. Thus they could be recovered by feeding them to an independent system that reruns the trigger software. In order to accomplish this reprocessing the L2 and EF algorithms are repackaged in a regular batch job.

It also makes sense to feed the *debug.HLTError* into the reprocessing, as some of the algorithm errors are due to missing fields in the event header. Those fields might be missing because of network timeouts which are transient errors and therefore recoverable. Sometimes it is possible to reconstruct the missing fields of event header making use of redundant information from other fields. In the end, both *event header analysis* and *reprocessing* run on every event of every debug sub-stream.

In order to profit from existing software we evaluated several batch-like production systems. We finally opted for Ganga[3], which is officially supported by ATLAS. It provides several built in back-ends and applications. It also provides persistency, and some general purpose utilities.

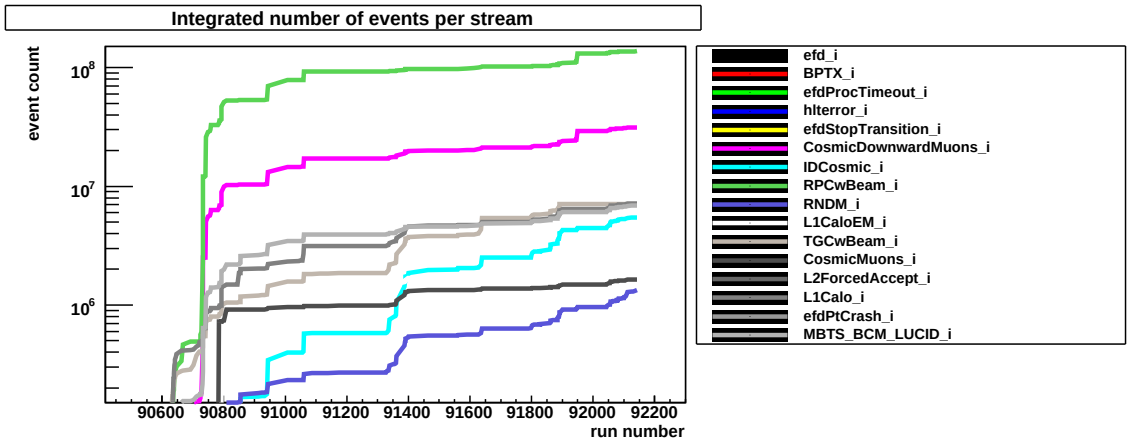

Figure 3: Integrated number of events per sub-stream in the period September-November 2008. The horizontal axis is the run number. Bigger run number means later in time.

However it still has a major drawback: it is oriented to user analysis, where the job is a monolithic piece that cannot be modified at runtime, and where no progress is reported until the full job ends. We created an extension of Ganga to solve this problem, Ganga Real-Time Extension. It is based on a hierarchical client server architecture, where a client is associated to each subjob. As the subjobs finish, they contact the server that appends the partial report to the overall status. This way the effect of progress update is accomplished. A higher level client-server layer is also available to provide extra flexibility. In the setup with the extra layer, all the servers from the first layer act as clients for a central server, creating a tree topology. This allows the application to define an appropriate granularity in terms of subjobs.

In the case of the TriggerOffline, during an ongoing run when the manager notices that new debug stream files are avaliable in Castor, it schedules a job that consists of several independent subjobs. In the next iteration, when more files are avaliable, a new job is scheduled. All the main jobs from the different iterations merge their results in the central server.

A major advantage of using Ganga is that it makes trivial to jump from using dedicated resources to using the Grid.

3. Cosmic Ray Runs

In the period August-November 2008 the TriggerOffline system was validated with cosmic ray runs. Figure 3 shows the integrated number of events per sub-stream during this period for about 100 different runs. It shows all three types, physics (*e.g.:* CosmicMuons, etc), calibration (inner detector, electromagnetic calorimeter, etc), and debug (efd, efdProcTimeout, L2ForcedAccept, hlterror, efdPtCrash). Note that the names vary slightly from the ones used in previous sections, but it is only a matter of nomenclature. Here *debug.HLTError* is called *hlterror*, *debug.L2Error* matches *L2ForcedAccept*, and *debug.EFError* is subdivided into several groups *efd*, *efdProcTimeOut*, *efdStopTransition*, *efdPtCrash*. The lines that correspond to the debug sub-streams do not show up in figure 3 because of the difference in magnitude. A zoom-in to the debug stream is

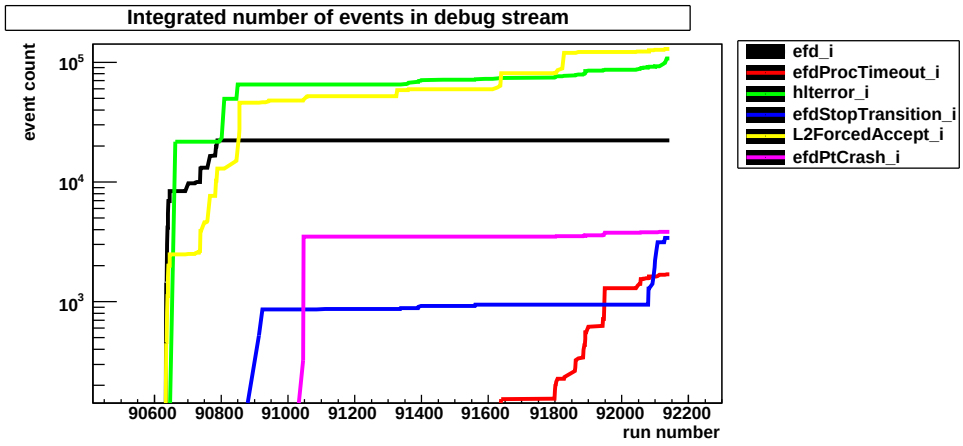

Figure 4: Integrated number of events per debug sub-stream in the period September-November 2008.

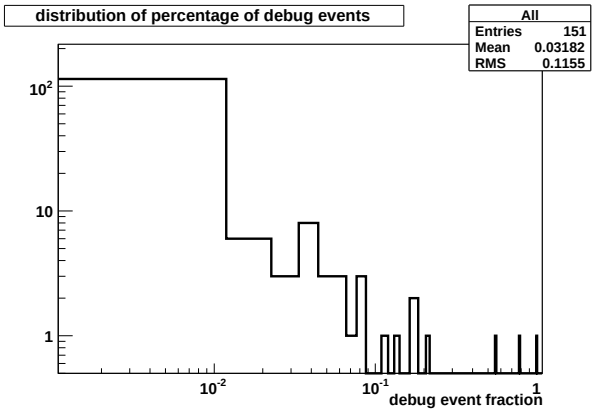

Figure 5: Distribution of the fraction of debug events for 151 cosmic runs.

provided in figure 4. By the end of this period of cosmic runs, the total number of events in the debug stream represents less than 0.2% of the total.

The distribution of the relative number of debug events to the total physics events is shown in figure 5. In the vast majority of the runs less than 1% of the events go into the debug stream. However, in longer runs the fraction reached over 30% (see figure 6). These short runs are mainly tests of new configurations where things did not turn out as expected. During long stable runs, the fraction of events in the debug stream is much lower.

One should note that TriggerOffline is mainly a debugging tool rather than a tool for recovering from online problems. When the TDAQ system experiences important problems, runs are stopped, analyzed, and once the problem is understood and tested (with the help of the TriggerOffline) the corresponding changes are applied to the data acquisition system. Nevertheless the TriggerOffline is able to recover some problematic events and reassign them to the physics streams.

4. Conclusions

When the ATLAS online system experiences software errors, its fault tolerance mechanism

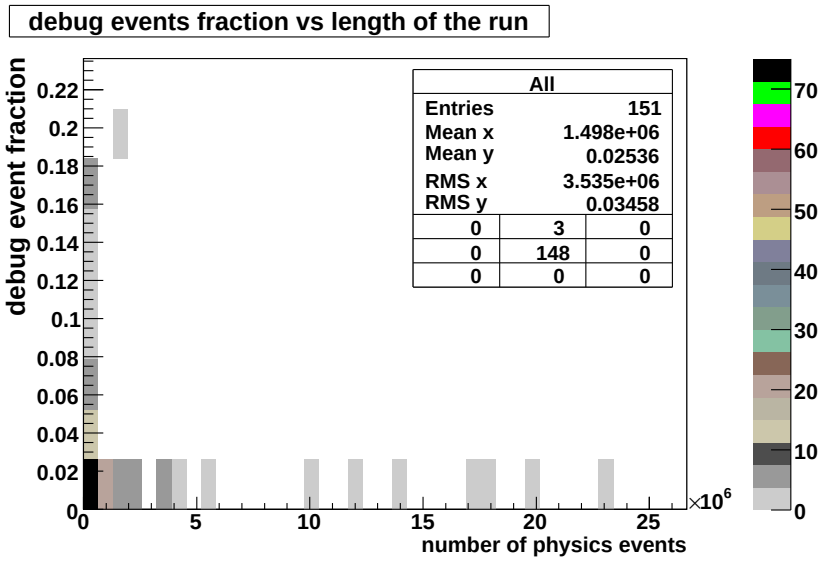

Figure 6: Correlation between the fraction of debug events with the lenght of the run for 151 cosmic runs.

routes the corresponding event data to the debug stream. We have developed a software package called OfflineTrigger that analyzes and tries to recover these events in quasi real-time. The automatic reports generated by the TriggerOffline provide feedback to the TDAQ experts, helping reduce the turn-around time for fixing online problems.

The TriggerOffline software has been helpfull in diagnosing problems during cosmic runs. Now, it is applied beyond its original scope, in validation of new trigger menus and releases, before they are comissioned in the TDAQ system.

References

[1] The ATLAS Collaboration. Atlas high-level trigger, data-acquisition and controls: Technical design report. CERN/LHCC/2003-022, 2003

[2] Jean-Phillipe Baud et al. CASTOR status and evolution, Computing in High Energy and Nuclear Physics, La Jolla California, March 2003

[3] Johannes Elmsheuser et al. Computing in High Energy and Nuclear Physics, September 2007, Victoria, Canada, September 2007