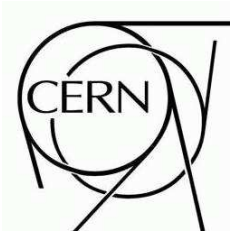
Draft version 1.0

ATLAS NOTE

ATL-INT/2009-000

April 13, 2009



Trigger Validation Dashboard: requirements, design and instructions

Alejandro Oyarzún, Will Brooks and Ricardo Gonçalo

Abstract

The trigger dashboard was designed to assist in the offline validation of the trigger software. The validation activity aims to guarantee the quality of the trigger software, including the level 1 simulation of the trigger response and the high-level trigger reconstruction and selection algorithms as well as configuration and steering infrastructure. The purpose of the trigger dashboard is to collect and display useful information to assess the performance and quality of the trigger software over long periods.

Contents

> ”A project is like a road trip. Some are like driving to the store in broad daylight. But most projects worth doing are more like driving a trruck off-road, in the mountains, at night. As a tester, you illuminate the road ahead so the programmers and managers, however they bicker over the map, can at least see where they are, what they’re about to run over, and how close they are to the cliff.” [1] C. Kaner, J. Bach, B. Pettichord

1 Introduction

The trigger validation activity [2] aims to guarantee the quality of the trigger software. Included in this domain are: the level 1 simulation of the trigger response, the high-level trigger reconstruction and selection algorithms, the configuration and steering infrastructure, and the trigger analysis tools. The purpose of the trigger dashboard is to assist in this activity by providing easy access to a series of software quality and performance metrics over long periods.

The trigger dashboard will automatically harvest quantities from a variety of nightly tests and store them in a database. Users will be able to query this database through a web-based interface to obtain and display these quantities and their evolution over time. This will give indications on the evolution of the software performance over time.

This note describes the trigger dashboard. Section A describes the requirements that were identified and that the tool should satisfy. Section 2 describes the overall design of the tool. Section B gives detailed instructions on the configuration and usage of the tool. We expect this tool to be valuable for trigger software validation and release coordination, as well as to debugging trigger and pinpointing when possible problems first occurred.

1.1 Trigger validation and automatic tests

The validation activity is carried out by a team spread around several continents. Any common tool therefore needs to be have a web interface. It should not require specialized knowledge for its use, since it should allow the validation activity to eventually become a non-expert task.

Much of the trigger validation daily activity consists in monitoring the result of a series of tests. These are performed nightly and are based on two test “scaffolds” or test infrastructures, common to other ATLAS software domains. The test scaffolds are known as ATN [3] (ATLAS Test Nightlies) and RTT [4] (Run Time Tester). ATN runs in the set of computers used for building the release every night and is managed by the NICOS system [5]. The ATN system can be used to perform relatively quick tests. These usually run Athena on a few events and take typically 15 minutes or less. RTT tests run on a dedicated set of machines at CERN. Typically these are used to perform longer tests on 1000 events or more. The major time constraint for both types of test is that the system must be free when the following nightly starts to be built or tested. Other kinds of test batteries exist but are not at the moment used systematically in the trigger validation.

Both ATN and RTT test results are displayed in dedicated web pages hosted at CERN. Test results are also copied to directories in `afs`. Results are kept during 7 days, corresponding to a full cycle of nightly releases (named `rel_0` to `rel_6`). Results older than a week are lost unless a record is kept independently of the test scaffolds. This is one of the purposes of the Trigger Dashboard.

In the following, a *test variable* will correspond to a given quantity (e.g. average time per event spent in trigger processing) measured by a given test (e.g. running the electron trigger selection on 1000 $t\bar{t}$ events). In some cases, priorities are assigned to a set of related requirements.

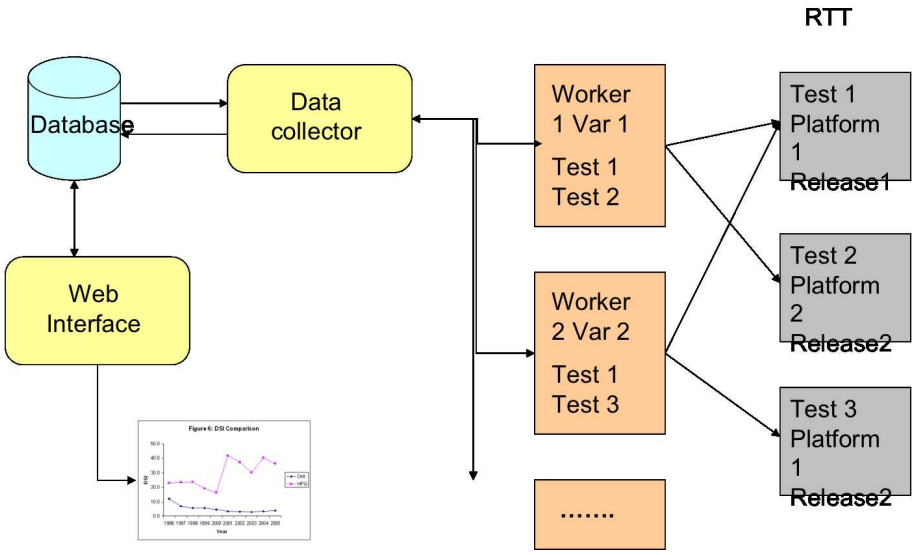


Figure 1: Overall design of the trigger dashboard.

2 Tool Design

Figure 1 illustrates the overall design. The tool is centered around a database implemented in a SQLite file. The database stores both the variables harvested from the nightly tests and the dashboard configuration. A "Data Collector" script collects test variables according to the tool configuration and returns their values to the database. The dynamic web interface, uses Python scripts and the Django library of web utilities to provide an administrative interface to the tool. A different web page allow users to query the variables stored in the database and to display them in tables and graphically using ROOT libaries.

2.1 Variable harvesting

Variable values are harvested every day by the Data Collector from the nightly test files left in afs. This module is a `bash` shell script which executes a series of other "worker" shell scripts, also represented in figure 1. These worker scripts are the ones which retrieve the variable values from the files left in afs by the nightly tests.

The data collector interacts with the database through command-line SQL queries to obtain the list of worker scripts to execute as well as their configuration. This configuration includes, for example, the complete path to the file containing the variable to be extracted. The worker scripts are written specifically to retrieve the value of a given variable or array of variables. The fact that they are simple shell scripts allows a great deal of flexibility. They can bery simply use command-line tools such as `grep`, or `awk`. They can also call other scripts or execute ROOT macros to extract the sought after values.

2.2 Data and configuration storage

The database schema is shown in figure 2. The name and description of each test variable is contained in the table named "Vars". Each variable is identified by a few additional parameters, such as the name of the test which produces this variable, the release and platform on which this test is executed. These parameters are contained in the table named "Vars_Date". The complete list of parameters identifying each variable value are:

- Suite: intended to identify the test suite or scaffold, e.g. `RTT` or `ATN`

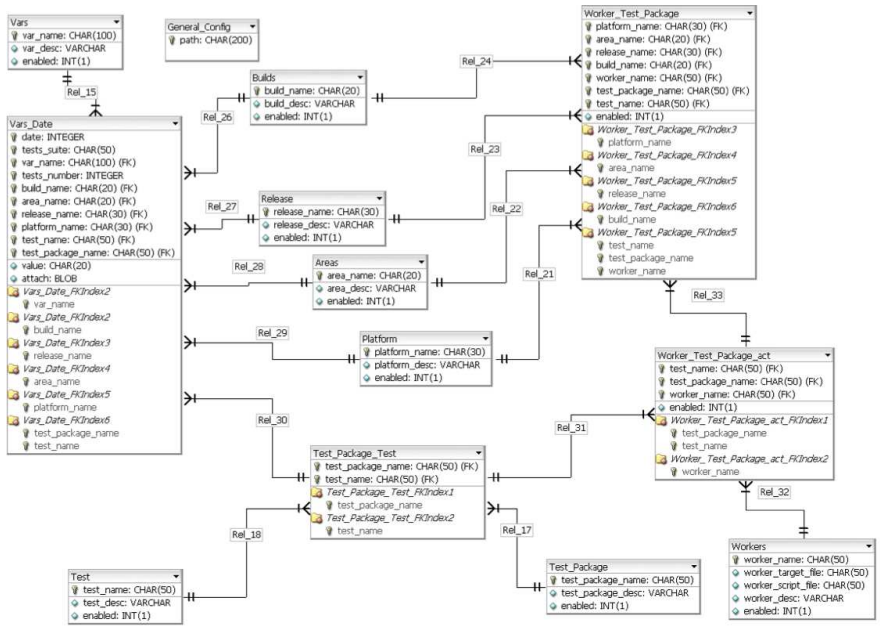


Figure 2: Schema of the Trigger Dashboard database

- Build: `opt` or `dbg` for optimised or debug builds
- Area: intended for future use, possibly as `offline` and `integration`
- Release: e.g. 15.3.0 for nightlies leading up to 15.3.0
- Platform: e.g. `i686-slc4-gcc34` for a build for Scientific Linux 4 using gcc 3.4
- Test name: test which produces the variable, such as `AthenaModernRDOtoESDAOD`
- Test package: software package used to run the test, e.g. `TrigAnalysisTest`

These parameters are used by the data collector to build the path to the file containing the variable to be extracted.

SQLite imposes some limitations in terms of concurrent data access and volume of data stored. These limits are, however, more than sufficient for the long-time operation of the tool. The volume of data to be stored is around 200 MB/year or less, and SQLite files are expected to take up to 2 GB.

2.3 Administrative interface

2.4 User interface

2.5 Implementation Details

The current version of the trigger dashboard was developed against the following versions of external software: Python 2.5; SQLite XXX, Django YYY; Apache module ZZZ.

Due to CERN rules regarding internet security, the tool is currently installed in a dedicated web-server machine, `voatlas24`, which also hosts other ATLAS components.

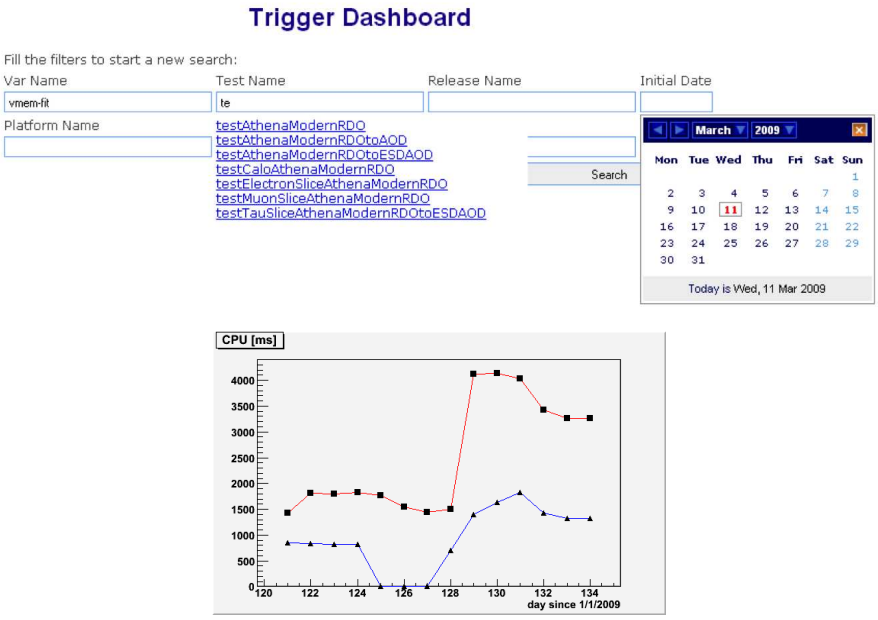
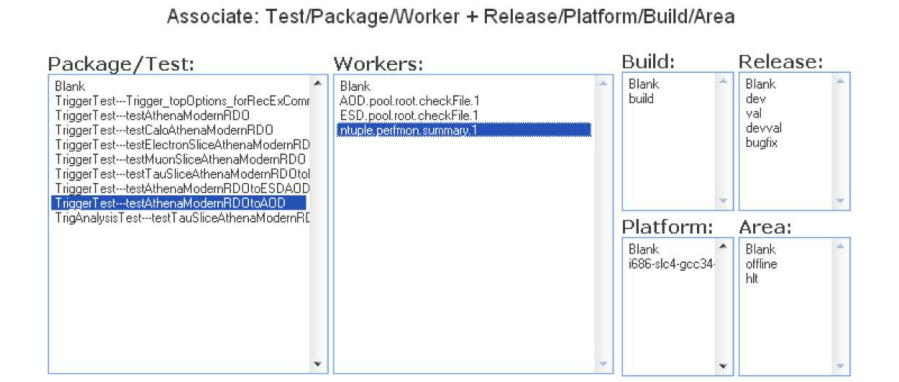


Figure 3: User interface of the Trigger Dashboard.



En/Dis	Package	Test	Worker	Build	Release	Platform	Area
1	TriggerTest	testAthenaModernRDOtoAOD	ntuple.perfmon.summary.1	build	dev	i686-slc4-gcc34-opt	offline
1	TriggerTest	testAthenaModernRDOtoAOD	ntuple.perfmon.summary.1	build	devval	i686-slc4-gcc34-opt	offline
1	TriggerTest	testAthenaModernRDOtoAOD	ntuple.perfmon.summary.1	build	dev	i686-slc4-gcc34-opt	hlt

Figure 4: Administrative interface of the Trigger Dashboard.

3 Conclusions

... conclusions

References

[1] C. Kaner, J. Bach, B. Pettichord, Lessons learned in software testing: a context-driven approach, (John Wiley & Sons, New York,).

[2] D. E. Ferreira de Lima et al., (PoS(ACAT08)084, ATL-COM-DAQ-2008-018, Erice, Italy, November 2009).

[3] ATLAS Test Nightlies and NIghtly COntrol System web pages, http://atlas-computing.web.cern.ch/atlas-computing/links/distDirectory/nightlies/global/nightly.html.

[4] Run-Time Tester web page, http://www.hep.ucl.ac.uk/atlas/AtlasTesting/.

[5] A. E. Undrus et al., (arXiv:hep-ex/0305087v1, La Jolla, California, USA, March 2003).

A Requirements

This section describes a set of requirements that guided the design of the trigger dashboard. The aim was to produce a useful validation tool that required little maintenance and was easy to use by non-experts.

In the following, a *test variable* will correspond to a given quantity (e.g. average time per event spent in trigger processing) measured by a given test (e.g. running the electron trigger selection on 1000 $t\bar{t}$ events). In some cases, priorities are assigned to a set of related requirements.

A.1 Validation requirements

To be a useful tool for the trigger validation, the trigger dashboard needs to fulfil a set of basic requirements:

1. The functionality provided by the tool must be accessible through a web interface, to provide easy access to collaborators around the world
2. The tool should require low maintenance when something changes in the trigger tests
3. The implementation should use easily available and public domain technology that requires minimal updates
4. The tool must allow access to test variables over a long period
5. The tool should automatically harvest test variables from nightly test results and be robust against sudden changes or failures in the tests

A.2 Tool Features

This section aims to identify features of the trigger dashboard that will help to fulfill the above requirements.

1. Storage: the values of test variables shall be kept indefinitely in storage and be classified according to unique criteria, including the test date, test name/identification, and a short textual description
2. Variable harvesting: the trigger dashboard shall automatically collect values of variables from RTT and ATN or other test scaffolds which write results to afs
3. Tool configuration: it shall be possible to configure the tool to collect new variables; this should require around one hour of work or less by an authorised expert
4. Access to information: it should be possible to access the test variables and perform complex queries on them through a web interface. As an example, it should be easy to see the evolution of a variable over time and correlations with other variables
5. Access control: it should not be possible to non-authorised people to change the tool configuration or modify any of the collected data

A.3 Functional Requirements

A.3.1 Variable Storing

1. Each variable shall correspond to a quantity extracted from a single test executed every night and should be kept independently of other variables, even if these have the same name

2. The technology used for storing variable values should be able to store of the order of 2000 floating-point quantities every day from each of about 4 build platforms; the overall volume increase expected is of around 200 MB/year or less

3. the variable values should be kept for at least 5 years

A.3.2 Variable Harvesting

1. The harvesting of test variable values shall be done automatically by the tool from files in afs

2. The tool shall be robust against missing files, directories, or the occurrence of the variable in the file

3. The tool should be able to harvest variables from text files, ROOT files, or using calls to the operating system

A.3.3 Tool Configuration

1. It should be possible to easily configure the Trigger Dashboard to retrieve a new variable or an existing variable produced by a new test

2. The tool configuration should be done through a web interface accessible through a password

3. It is not foreseen that more than one authorised user will simultaneously edit the tool configuration

A.3.4 Access to Information

1. A non-expert user should be able to query the test variables stored and display graphically their evolution over time

2. The exact values of test variables should be accessible in table format

3. It is not expected that many users will simultaneously access the trigger dashboard; as an estimate, the numbers of simultaneous users should be always less than 10 and usually only one

A.4 Non-Functional Requirements

B Instructions

B.1 User Interface

B.1.1 Displaying a monitored test variable

... between certain dates
... other types of queries

B.2 Administrative Interface

B.2.1 Logging in

B.2.2 Adding a new test variable to be monitored

B.2.3 Enabling/disabling a test

... etc