CONTROL AND MONITORING OF ON-LINE TRIGGER ALGORITHMS USING A SCADA SYSTEM

E. van Herwijnen, L. Abadie, A. Barczyk, B. Damodaran, M. Frank, B. Gaidioz, C. Gaspar, R. Jacobsson, B. Jost, N. Neufeld, CERN, Geneva, Switzerland

F. Bonifazi, Istituto Nazionale di Fisica Nucleare (INFN), I-40127 Bologna, Italia

O.Callot, Laboratoire de l'Accélérateur Linéaire (LAL), F-91898 Orsay, France

H. Lopes, Instituto di Fisica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brasil

Abstract

LHCb [1] has an integrated Experiment Control System (ECS) [2], based on the commercial SCADA system PVSS [3]. The novelty of this approach is that, in addition to the usual control and monitoring of experimental equipment, it provides control and monitoring for software processes, namely the on-line trigger algorithms.

Algorithms based on Gaudi [4] (the LHCb software framework) compute the trigger decisions on an event filter farm of around 2000 PCs. Gaucho [5], the GAUdi Component Helping Online, was developed to allow the control and monitoring of Gaudi algorithms. Using Gaucho, algorithms can be monitored from the run control system provided by the ECS. To achieve this, Gaucho implements a hierarchical control system using Finite State Machines.

In this article we describe the Gaucho architecture, the experience of monitoring a large number of software processes and some requirements for future extensions.

THE PROBLEM

The configuration of the LHCb DAQ is shown in Figure 1. The data from the detector is sent from the front end electronics via the readout network to the event filter farm. The event filter farm consists of the order of 50 subfarms with approximately 35 dual CPU dual core nodes each. The data acquisition is distributed across the many nodes of this large event filter farm. Each node processes a fraction of the events from the detector. The configuration and control of this farm is a complex task which is done by the experiment's control system. In addition to controlling and monitoring of the hardware, the ECS manages the trigger processes running on the event filter farm. This means the ECS sends commands to the processes to control their activities and it receives counters, rates, histograms, status variables and messages for display. Furthermore, the ECS needs to be the central point where the monitoring data from all the processes is collected, so that full histograms are produced and tagged regularly for further analysis.

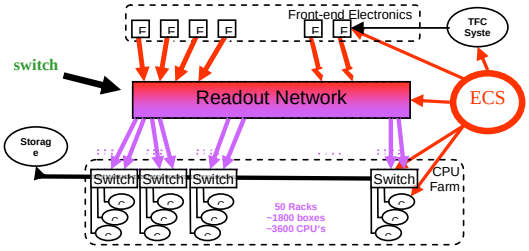

Figure 1: Configuration of the LHCb DAQ

THE SOLUTION

To obtain a seamless integration with the overall run control system, we use PVSS to monitor the on-line trigger algorithms. The configuration, starting and stopping of the algorithms is done using the run control system's PVSS user interface. Counters, rates, status variables and messages are displayed in PVSS panels. The data from the trigger algorithms (counters as well as histograms) are accessed via the communications package DIM [6]. The manipulation of histograms will take place by a ROOT [7] based histogram framework, which will also handle automatic summing of histograms.

GAUCHO

The Gaucho system consists of:

- A C++ part integrated into Gaudi allowing the trigger algorithms to publish monitorable objects
- The communications package DIM for publishing/subscribing to data upon change or at regular intervals
- A PVSS backend for displaying monitored objects.

C++ part

The Gaucho C++ package implements the monitor service interface (IMonitorSvc) as DIM services and RPCs. It has one method called DeclareInfo to declare variables and histograms for monitoring. Algorithm writers use it to publish the quantities that need to be monitored by the ECS. Special care was taken to ensure the same code can be used without modifications in both the online and offline contexts.

Messages are published by the OnlineMsgSvc (in the offline context by the MsgSvc).

A Gaudi shell instantiates a DIM server and creates an FSM machine allowing the algorithms to start and stop. The state (not_ready, ready, running, error) is published via a status variable (see Table 1). The Gaudi shell also instantiates the Gaudi Application Manager. This is a flexible and generic way to add a FSM to any Gaudi algorithm. Indeed it is planned that, in addition to the trigger algorithms, the event building software and various other Data Acquisition programs handling the data will also be constructed in this way so that they can all be controlled by the farm's run control system.

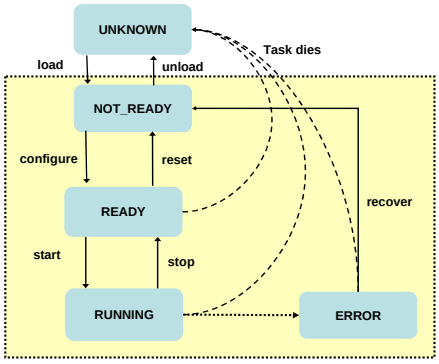

FIgure 2: The FSM implemented by the Gaudi shell

The MonitorSvc is currently used by trigger algorithm developers. Information is only sent when subscribed to, so there is no performance penalty when the algorithms are run offline.

DIM

The communications package DIM provides the connection between the Gaudi algorithms and PVSS.

The correspondence between the datatype of the published object and the DIM service is shown in Table 1.

Table 1: Object types and DIM commands

Object Data type	**DIM Method**
int	DimService(infoname.c_str(), (int&) var)
double	DimService(infoname.c_str(), (double&) var)
string	DimService(infoname.c_str(), (char*) var_cstr())
AIDA::IHistogram	DimRpc(histoname.c_str(), "I:1","F")

PVSS backend

Using the PVSS user interface (the run control), algorithms can be stopped/started and counters and histograms can be followed in real-time. The results are combined at the level of nodes, subfarms and the full farm, so that it is easy to verify the correct functioning of the trigger.

The PVSS DIM client is a convenient tool for subscribing to counters and histograms published by Gaudi algorithms using the MonitorSvc. The subscribed data can be viewed using a three tier panel hierarchy, i.e. per algorithm, summed per node and summed per subfarm (currently only one subfarm is implemented). The three tier panel hierarchy is shown in Figure 2.

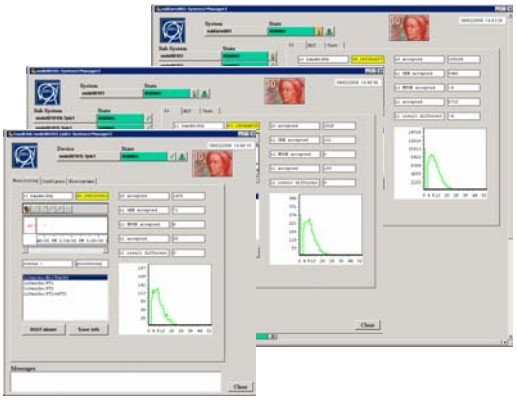

Figure 3: Three tier Gaucho PVSS panel hierarchy

The configuration of the farm node structure in PVSS, i.e. the number of subfarms, the number of nodes in a subfarm, the number and types of algorithms running on each node is done independently of the Gaucho system. The CERN Joint Controls Project (JCOP [8]) has a framework of PVSS applications for the LHC experiments. In this context, a PVSS tool called fwEFF for the configuration of event filter farms is developed.

After configuring the farm with fwEFF, the corresponding Gaucho PVSS structure is created by a script.

Gaucho offers the flexibility to monitor only counters, or both counters and histograms.

SIZE OF MONITORING DATA

To evaluate the load of Gaucho on the network we have calculated the size of the monitoring data. This is shown in Table 2.

Table 2: Typical size of monitoring data

Item	Test algorithm	Typical Trigger algorithm
# strings	1 status, 5 comments	1 status, 3 comments
Bytes for strings	228	188
# counters	4 ints, 1 long	3 ints
Bytes for counters	20	12
# histograms (nbins)	4 1D (5,80,60,60), 1 2D (100)	24 1D (4x10, 1x11, 1x20, 2x40, 12x80, 4x150) 2 2D (200)
Bytes for histograms	1220	10044
Total #kb	1.5	10.5

Dataflow into PVSS

Tests were done on a testbed event filter farm configured as a single subfarm with 40 nodes (dual CPU). 6 jobs were run on each node (2 test algorithms, 4 trigger algorithms). The system was setup such that the data was published every 20 seconds. This amounts to a maximum of 1.6 Mbytes being sent every 20 seconds, out of which 4 kbytes is from counters, the rest from histograms. This quantity of data is well within the capacity of the network (12 Mb/s or 100 Mbits/s).

SYSTEM PERFORMANCE

In our tests with 240 jobs, PVSS was running on a Windows Xeon 3GHz CPU with 2 GB RAM.

- For counters (~1200), the summed values were displayed on the panels via PVSS "datapoint functions", updated every 20 seconds. The PVSS system used 3-12% of the CPU and 700 Mb memory.
- Histograms (~3000) were updated and summed sequentially, which meant once every 4 minutes. The PVSS system used 5-55% of the CPU and 700 Mb memory.

HISTOGRAMMING FRAMEWORK

The experience of displaying counters and histograms in PVSS highlighted our need for a more separated histogramming framework. This work is now ongoing. Some of the requirements and components of this framework are: histogram booking using a database, histogram access should be possible for online and offline use, a flexible histogram presenter integrated with the run control, and regular saving.

Histogram "adders" for accessing histograms

For example, the cascaded summing of histograms could be implemented with stand alone "histogram adders", rather than inside PVSS (see Figure 4).

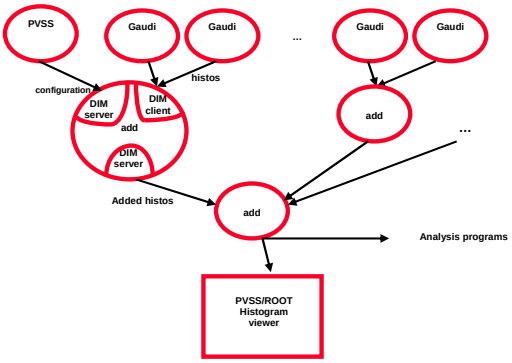

Figure 4: Adders for histograms

The adders perform the following tasks:

- Find out from PVSS which histograms should be added (configuration).
- Subscribe directly to the Gaudi jobs as a DIM client.
- Collect the histograms from the Gaudi jobs and sum them.
- Publish summed histograms (using the MonitorSvc) to clients such as the histogram presenter or an analysis program or to subsequent adders.

The adder can be implemented as a Gaudi algorithm with the signature as shown in Table 3.

Table 3: Signature of the histogram adder

Function	description
adder(std::string name, std::vector <std::string>)	The constructor creates a DimCommand with the name of the node, subfarm or farm and a vector containing names of histograms to be added

If the adder is run inside the Gaudi FSM shell mentioned above, it can be controlled by the run control system.

Access to summed and individual histograms will be for display (online) and for regular saving (offline); the regular saving can be done by a separate stand alone program or inside the adders. The problem of saving the

final version of histograms before reset needs to be addressed.

There is also a requirement to inform clients when new versions of histograms are available and to "push" them to clients; it is not clear if the current implementation of the MonitorSvc which uses a "pull" mechanism implemented with a DimRpc (see Table 1) is satisfactory.

Histogram presenter

The histogram presenter should provide well defined display pages allowing a selection of histograms (probably via PVSS). As ROOT is the standard histogramming package for physics analysis the histogram presenter will be based on ROOT. All the interactive features to manipulate the histograms in ROOT should be available from the histogram presenter. Examples of local manipulations on displayed histograms include changing the scale, fitting, zooming and changing the camera position for lego plots.

Some of the monitoring information will be in the form of PVSS trend plots; it is desirable to have a single display containing these trend plots together with ROOT histograms. This may be possible in the next release of PVSS which will use the Qt [9] drawing system that is also used by ROOT.

Finally the histogram presenter should have all the features that one expects of a system that allows the shift operator to understand the quality of the recorded data. For example, we need a mechanism to analyse a given histogram independently without interfering with the main window, in which it is continuously updated; it should be possible to compare with reference plots.

Current ROOT viewer

In our current implementation of Gaucho, we have a ROOT viewer that implements a DIM client to display 1D and 2D histograms (see Figure 5). This viewer allows us to save data on demand or at regular intervals as text files (for counters) or as root files (for histograms). It could be extended to accommodate the requirements mentioned in the previous section.

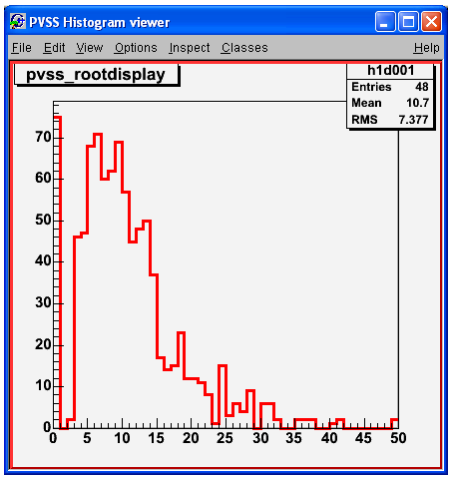

Figure 5: ROOT display

CONCLUSIONS

We have shown that it is possible to monitor counters from trigger algorithms using PVSS, a commercial SCADA system. A ROOT based histogram framework is planned. Histogram adders controlled by PVSS will connect directly to Gaudi jobs, and send their results to other programs for further analysis, or feed a viewer. A histogram viewer is planned in PVSS/ROOT for easy viewing/saving of histograms. A histogram database will contain the details of histograms for quality analysis.

Deleted: QT

REFERENCES

[1] http://lhcb.cern.ch/.

[2] http://lhcb-online.web.cern.ch/lhcb-online/ecs/default.htm/.

[3] ETM, "PVSS II, Version 3.0", Eisenstadt, Austria, 2004.

[4] G. Barrand et al., "GAUDI – A software architecture and framework for building LHCb data processing applications", Proc. of CHEP, Padova, Italy, February 2000.

[5] P. Vannerem, "Distributed Control and Monitoring of High-Level Trigger Processes on the LHCb on-line Farm", ICALEPS 2003, Gyeongiu, Korea, October 2003.

[6] C. Gaspar, "DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication", Proc. of CHEP, Padova, Italy, February 2000.

[7] http://root.cern.ch/.

[8] IT-COBE group, "JCOP framework, version 2.3.3", CERN, Geneva, 2005.

[9] Trolltech, Cross Platform C++ application framework http://www.trolltech.com/products/qt/index.html