Automated Data Quality Monitoring in the BaBar Online and Offline Systems

S. Metzler[1], E. Chen[1], G. Dubois-Felsmann[1], P. Bright-Thomas[2] for the BaBar Computing Group

[1] California Institute of Technology, Pasadena, CA, USA
[2] University of Birmingham, Birmingham, UK

Abstract

An automatic monitoring system, developed for the BaBar experiment, allows for the evaluation of the contents of data quality diagnostic histograms produced in other components of the experiment. Both the data acquisition system and the subsequent reconstruction make use of this facility.

It supports periodic statistical tests of histogram bin contents against reference distributions, which may be other stored histograms or analytic distributions, and tests the parameters resulting from fits to histograms for consistency. Tests may be performed against entire one- and two-dimensional histograms, or on a bin-by-bin basis, a capability used for monitoring the appearance of single dead or hot channels in the detector. Fitting of histogram data to specified functional forms allows testing of parameters against reference values.

Alarms of various sorts may be logged and sent to operators at consoles or to experts via e-mail, based on configurable tolerance thresholds for each test performed.

Graphical tools are available for configuring and monitoring the system. The tools allow performance monitoring of the system and access to the data underlying the statistical tests.

Keywords: Data Acquisition, Data Analysis, Online

1 Introduction

The BaBar online system's Online Event Processing component [1] performs software triggering and online data quality monitoring on a farm of 32 Sun Ultra–5 workstations. The event rate delivered to this system from the event builder may range up to 2000 Hz, and has averaged approximately 1000 Hz in BaBar running through December 1999. The software trigger is required to reduce the event rate to 100 Hz or less. The online system is expected to provide real-time access to this data for monitoring purposes.

Diagnostic data is produced on all 32 computing nodes. The input rate for monitoring can be controlled to give full or sampling access, to all input events or only those passing the software trigger, depending on the need for statistics, the need for certain raw data quantities, and the performance of the monitoring code. Diagnostic data is produced in monitoring code using the Distributed Histogramming Package [2], which allows data summed over all nodes, or for any individual node, to be queried on demand.

Thousands of diagnostic objects are produced for each run. It is impossible for the shift crew to monitor all of this information. It is typical for detector subsystems to produce a small number of high-level displays that the shift crew checks every run. It is important to have automatic monitoring in order to check the integrity of the entire system.

2 Diagnostic Data Types

Histograms and “scalers” are provided as the primary diagnostic data types. Histograms provide data that are integrated over time. “Scalers”, in our terminology, provide tracking of quantities as a function of time.

There are three different types of histograms. Users can select from one dimensional histograms, two dimensional histograms, and one dimensional profile histograms.

Scalers come in four varieties: averaging, integrating, value and multi. Each tracks a specified quantity over a history of specifiable time intervals. Entry of data into a scaler is decoupled from the specification of the time interval boundaries, to allow enforcement of common set of time intervals across many independently accumulated scalers. An averaging scaler’s value for an interval is the weighted average of all accumulated entries during that interval. When combined across multiple nodes, the nodes’ interval values are averaged. Integrating scalers have a value that is the sum of all entries during an interval. When combined across nodes, the values are summed. Value scalers retain the latest entry in each time interval. When combined across nodes, a single node’s value is used as the result. Multiscalers are composed of the above simple types. Their combination semantics depend on the types of single scalers contained.

3 Comparison Techniques

Each automated comparison is defined in a comparison record object. Each comparison record contains information regarding which histogram to test, the type of test, what responses should occur, and the frequency of comparison.

Typically, the user creates a text file representation of his or her comparison records, which are then parsed by the monitoring application and converted into comparison record objects. The monitoring application then executes these comparison records based on their specified frequency.

3.1 Comparisons against Fixed Spectrums

The user can compare a live histogram with a reference histogram or a function spectrum, such as a Gaussian. The statistical tests which can be used are Kolmogorov-Smirnov and χ^2. Instead of a statistical test, the user can choose to perform a bin-by-bin occupancy test, where individual bins are checked to see if they change status from reference to test histogram. This is useful, for example, when users want to be alerted when channels go dead or hot.

3.2 Fitting

Fitting is needed to allow for variations in conditions that are difficult to handle with fixed spectrums. Histograms will be passed to a fitter and fit parameters will be returned. A specified subset of the fit parameters will be compared against allowed ranges and responses will be executed, if appropriate. We expect the ability to ignore some parameters to be very useful for dealing with unknown conditions, such as fluctuating background levels.

3.3 Scaler Comparisons

Scaler comparisons are not yet available in automatic monitoring. Scalers are presently checked only by eye. Comparisons against fixed values and as a function of other scalers are envisioned. This will allow checking quantities as a function of luminosity, for example.

4 Responding to Problems

For each comparison record, the user may specify associated responses. Each response is defined by a minimum and maximum confidence level, as well as a message and a message transport mechanism. If the confidence level returned by the comparison lies within the minimum and maximum specified by the response, then the response will be executed. In the case of the bin-by-bin comparison test, where the confidence level is meaningless, the alarm response will instead return a message that, depending on user choice, either summarizes how many bins had problems or states specifically which individual bins had problems.

Currently, all responses log error messages, but we expect more sophisticated responses as we gain experience. Sending errors to an email address, which may be used to send a page, as a stream and to our occurrence logger, which is an application of the CMLOG system [3] from the Thomas Jefferson National Accelerator Facility, are the three presently available message transports. The user can add as many alarm responses as desired.

5 Graphical Tools

Access to the data is essential for effective monitoring. We provide that access using CORBA [4]. We plan on providing a connection through Java Analysis Studio (JAS) [5] to help integrate data monitoring.

The automatic monitoring process presently has a custom Java *graphical user interface* (GUI). It displays monitoring statistics and provides administrative control over monitoring. Access to online comparisons is also available.

An error-browsing application provides a direct connection to automatic monitoring data. This application is a Java GUI that receives error messages sent to CMLOG. It identifies messages that originate from the automatic monitor and displays them. It then gives users a direct connection back to the automatic monitor and allows shift-takers to more closely inspect potential problems.

Integration of the above tools into a single application is important for the long-term effectiveness of automatic monitoring. We plan to migrate these tools into a single view and have that view available within JAS.

6 Other Uses

The automatic monitoring code is available in BaBar analysis framework modules so that it can be used within the standard BaBar application environment. This readily gives us the ability to set configuration parameters using TCL scripts. Further, it allows automatic comparisons to be performed within analysis code.

Online automatic monitoring is a framework application. This application is configured and executed using a TCL file. The framework is also used to provide a looping mechanism.

After the BaBar data has been reconstructed by the Prompt Reconstruction [6] system, the data will be validated using the automatic monitoring system with a single pass over all the diagnostic data. Configuration and error handling will utilize the existing monitoring functionality.

7 Construction Lessons

There were many valuable lessons to be learned from our experience with automatic monitoring. Some lessons relate to Object-Oriented design principles. Others are more a statement of the psychology of our users.

Flexibility in the types of comparisons that can be performed is an important feature that has been available from the beginning. We used abstraction to insulate the code from all the possible types of comparisons that were developed. There were several requests for new comparison types late in the development. They were easily accommodated within the structure without re-design.

Early prototyping is an important concept that we did not follow. We paid the price for this by not realizing early enough the difficulty users would have in configuring monitoring for their detector subsystems. Furthermore, users did not develop familiarity early on with the software and the mind-set of how it should be used with their system. This has made it more difficult than it should have been to have users adopt and use the system.

Insulation of software components was particularly important in this project because of the large number of dependencies on other BaBar software. Fortunately, many of the pieces were, in retrospect, well abstracted and provided not only insulation, but generalization. For example, we abstracted the concept of the source of our histograms. This allowed us to generalize our solution to handle not only data retrieved through CORBA, but also data retrieved using smart pointers within a single process and from legacy HBOOK histogram files. This has proved to be a very useful feature.

8 Conclusions

In this project we have re-learned some Object-Oriented lessons. Flexibility, insulation and prototyping are important pieces of a successful software project. In particular, we learned that we did not anticipate our users needs as well as we should have because the lack of early prototypes precluded timely feedback on the operation of the system. However, we provided a flexible system that has proven to be well insulated from the types of changes that have occurred through the duration of the project.

We have a working but incomplete system available now. Fitting has not been implemented and the generalization to scaler comparisons has not yet been finished. However, our first priority has been to provide a system that monitors histograms against fixed references. Our system has reached that goal. In summary, our system does provide automation of histogram comparisons, tools to access them and is in use within BaBar .

References

1 G. P. Dubois-Felsmann *et. al*, “The BaBar Online Computing System”, CHEP2000, Padova, Winter 2000.

2 S. Metzler *et. al*, “Distributed Histogramming”, CHEP’98, Chicago, Autumn 1998.

3 J. Chen *et. al*, “CMLOG: A Common Message Logging System”, ICALEPCS’97, Beijing, Autumn 1997.

4 S. Metzler *et. al*, “Production experience with CORBA in the BaBar experiment”, CHEP2000, Padova, Winter 2000.

5 T. Johnson *et. al*, “Java Analysis Studio”, CHEP’98, Chicago, Autumn 1998.

6 T. Glanzman *et. al*, “BaBar Prompt Reconstruction”, CHEP’98, Chicago, Autumn 1998.