Data Quality Monitoring Framework for the ATLAS Experiment at the LHC

A. Corso-Radu, S. Kolos, University of California, Irvine, USA

H. Hadavand, R. Kehoe, Southern Methodist University, Dallas, USA

M. Hauschild, CERN, Geneva, Switzerland

***Abstract* - Data Quality Monitoring (DQM) is an important and integral part of the data taking process of HEP experiments. DQM involves automated analysis of monitoring data through user-defined algorithms and relaying the summary of the analysis results while data is being processed. When DQM occurs in the online environment, it provides the shifter with current run information that can be used to overcome problems early on. During the offline reconstruction, more complex analysis of physics quantities is performed by DQM, and the results are used to assess the quality of the reconstructed data. The ATLAS Data Quality Monitoring Framework (DQMF) is a distributed software system providing DQM functionality in the online environment. The DQMF has a scalable architecture achieved by distributing execution of the analysis algorithms over a configurable number of DQMF agents running on different nodes connected over the network. The core part of the DQMF is designed to only have dependence on software that is common between online and offline (such as ROOT [1]) and therefore is used in the offline framework as well. This paper describes the main requirements, the architectural design, and the implementation of the DQMF.**

I. Introduction

ATLAS is one of the four experiments at the Large Hadron Collider (LHC) [2] at CERN. This experiment has been designed to study a large range of physics including searches for previously unobserved phenomenon such as the Higgs Boson and super-symmetry [3].

At about 140 million electronic channels and an event rate of 10^5 Hz it is essential to monitor the status of the ATLAS hardware and determine the quality of the data being taken in an efficient manner. In the online environment, this information can flag the shifter to take action to prevent taking faulty data. In the offline environment, one can perform more complex checks of the data to determine the quality of the data for various physics groups. Careful monitoring of data both online and offline is especially important at the beginning of an experiment where the environment is new and requires some experience to fully comprehend. Therefore having a common, user friendly tool can help the experiment quickly determine problems and then proceed to solve them efficiently. Experience from the Tevatron experiments has shown us that an automatic checking of data is needed in the complex physics environment of proton-proton collisions. DQMF has been designed to fulfill these requirements.

II. General Description

The purpose of the Data Quality Monitoring Framework (DQMF) is to apply analysis algorithms to various types of online monitoring data (histograms, messages, counters, etc.) according to a particular configuration, which is defined by detector experts. The results of this analysis may generate alarms when deviations from the standard are encountered. A summary of these results will be displayed to the shifter and will also be archived for future retrieval. Using this summary, the shift operator will make a final data quality assessment for a given run. The archived results will permit a check or refinement of this assessment offline.

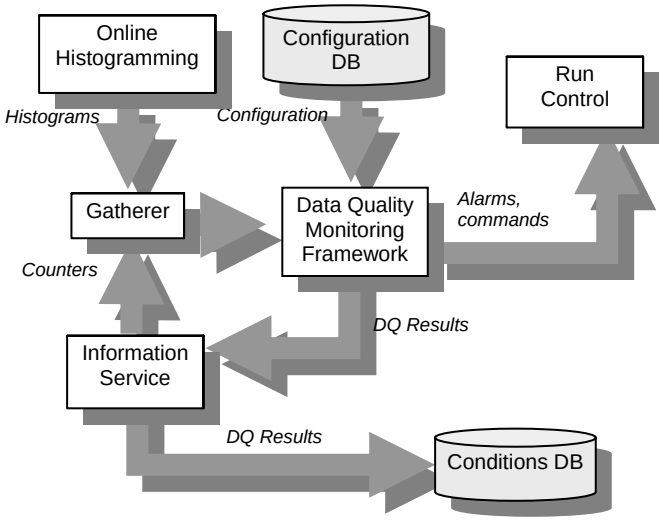

Fig.1: DQMF interaction with the online services.

DQMF interacts with the Online Monitoring Services as well as with some other Online Services, provided as part of the ATLAS TDAQ [4] software infrastructure (Fig. 1), in order to be able to fulfill its objectives, in particular:

- Information Service (IS) [5] is used to retrieve the required monitoring data and also to make the DQ analysis results publicly available.

- Online Histogramming Service (OH) is used to retrieve histograms produced in the current run and to transmit requests to histogram providers. Histograms generated within DQM algorithms are transmitted to OH to be displayed and/or archived.
- Gatherer is used to sum up histograms from different nodes. DQMF will then perform algorithms on full statistics histograms produced from the Gatherer [6].
- Error/Message Reporting Service (ERS/MRS) is used to send messages to the TDAQ system.
- Controls Service (CS) [7] is used to send messages to the TDAQ control system in case a run needs to be paused, reconfigured, etc. in order to avoid taking faulty data.
- Configuration Database (ConfDB) is used to store and retrieve all the information that is necessary to configure the DQM activity for the current run, i.e. algorithms to be used, monitoring data to be retrieved, etc.
- Conditions Database is used to store and retrieve DQ results.

III. Main Requirements

Since the DQMF will be used in the prompt analysis of data it must be able to analyze the input data in an efficient manner. Therefore the algorithms written for analysis must be light weight and must respect the time limits set for processing. The timing of the system and synchronization is important for keeping a consistent view of the status of the detector.

The algorithms written for the DQMF should be easily developed and tested in an offline manner by sub-component experts. This ensures that the results of the DQMF will be reliable and easily reproduced by the expert given the list of parameters used.

The DQMF must be portable to both Point1, online, and Tier-0, offline first pass reconstruction, running environment. It is envisioned that the DQM status from express streams, fast Tier-0 processing on subset of data, can be viewable at Point1 via a GUI. Although there may not be direct relaying of information from Tier-0 to Point1, the information from both environments shall be available to the shift personnel.

When possible, the DQMF implementation, utilizes the software tools which are already available. For example for archiving histograms and values produced by the DQMF we will use a combination of Monitoring Data Archiver (MDA) [8] and Online Asynchronous Interface to COOL (ONASIC) [9]. For performing fits within the algorithms we use ROOT[1]. For the configuration of the DQMF we use the Object Kernel System (OKS) [10], which provides object-oriented information representation.

IV. Architectural Design

At high-level the DQMF can be split into several sub-components as shown in Fig. 2. A detailed description of each component is given after the diagram.

A. DQM Database

This component contains a complete description of the DQMF for a particular DAQ configuration. This description defines where the DQM Agents (application that implements the DQMF) will be running, what information they will use, what algorithms they will apply to analyze this information, etc.

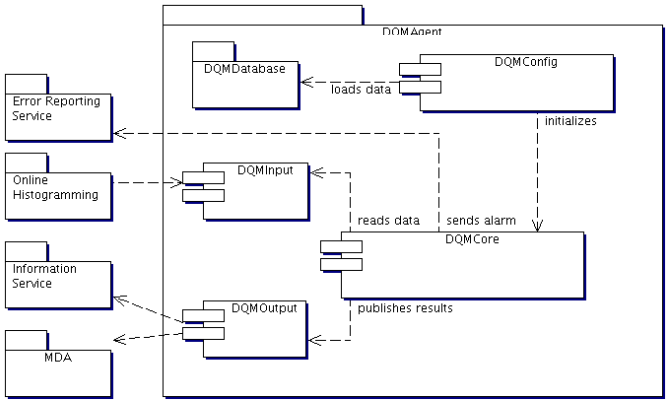

Fig. 2: Main components of the DQMF and their relationship with the online services.

B. DQM Database

This component contains a complete description of the DQMF for a particular DAQ configuration. This description defines where the DQM Agents (application that implements the DQMF) will be running, what information they will use, what algorithms they will apply to analyze this information, etc.

C. DQM Config

This component reads the DQM configuration corresponding to a DQM Agent from the DQM Database and makes this information available to the DQM Core. The main aim of this component is to hide the particular database technology that is used for the DQM Database implementation.

D. DQM Input and DQM Output

These are two generic interfaces that have specific implementations depending on the environment in which the DQMF is used. For the online usage they are implemented using the online services like IS and OH. In the offline environment one reads the monitoring information from a ROOT file. The offline results should then be sent to a database.

E. DQM Core

This component is a central part of the DQMF. It reads the input data via the DQM Input interface, performs analysis of these data and publishes the DQM Results via the DQM Output interface. In addition, it may generate alarms if the DQM Results deviate significantly from the standard values. The main components of the DQM Core are described below.

1) DQM Algorithm

The DQM Algorithm class provides generic interface for any kind of operations that can be applied to the monitoring

data in order to evaluate a status of the corresponding DAQ system elements. The algorithms work in a plug-in manner into the core. This allows user defined algorithms to be used in the system on the fly without modifying the core software.

DQMF provides a number of predefined DQM Algorithms for the most common operations like histogram comparison, histogram fitting, thresholds application, etc. In addition to the predefined ones, a user can develop his/her own DQM Algorithms.

A DQM Algorithm is responsible for producing the DQM Result object. In addition a DQM Algorithm can also produce some additional information during its execution such numerical results like fit results or a difference histogram. This information can be published to the Information Service and/or Histogramming Service in order to be used for eventual future analysis.

2) DQM Parameter

A DQM Parameter object represents the element of the DAQ system whose state can be assessed using a single monitoring information, i.e. a particular histogram or information object.

A DQM Parameter object is responsible for applying the algorithm defined for that object to a particular piece of information with the aim of producing the DQ status for the corresponding DAQ element. The description of the DQM Parameter is taken from the DQM Database and provides the following information:

- the location of the monitoring information that represents the state of a particular DAQ element
- the weight (or degree of importance when calculating the DQM Result) for that DQM Parameter
- the DQM Algorithm that has to be used for evaluating the DQ status of that element
- specific parameters for the given DQM Algorithm, e.g. fit parameters, minimum statistics, etc.
- reference values or histograms
- the list of thresholds values corresponding to Red/Yellow/Green DQM Result
- actions that have to be taken depending on the results of the DQM Algorithm execution.

3) DQM Region

An object of the DQM Region class represents a certain subset of the DAQ system elements and contains a set of DQM Parameters or low-level DQM Regions, which correspond to these elements. This class is used for representing a self-contained part of the DAQ system, like a detector, a sub-detector, a sub-farm, etc. For any particular DAQ partition all the available DQM Regions and DQM Parameters will be organized into a tree (DQ Tree) with the root node representing the state of the whole partition. This state is defined by the DQM Result calculated as a combination of the DQM Results for the contained DQM Parameters and DQM Regions. The rules for DQM Result calculation (for a given DQM Region) are provided by the DQM SummaryMaker that is associated with that region. A possibility to define different summary algorithms for different DQM Regions will also be supported. This class provides a reference to the DQM Summary Maker algorithm that produces the DQM Result for that region.

4) DQM Result

A DQM Result object represents the DQ status for a given DAQ system element. Each DQM Result contains a color code value (Red-Yellow-Green) that shows the state of the particular DAQ element. In addition to that it may also contain a reference to the detailed numerical results or histograms that have been produced by the DQM Algorithm.

5) DQM SummaryMaker

The DQM SummaryMaker is a special implementation of the DQM Algorithm interface that evaluates the DQM Result for a given DQM Region. To calculate a new DQM Result the DQM SummaryMaker object uses the DQM Results produced by the DQM Regions and DQM Parameters that belong to the given region.

V. Scalability

A. DQ Tree

In order to make the DQMF scalable its functionality is provided by a set of processes with each of them hosting an instance of the DQM Agent application. Each DQM Agent is responsible for the subset of all the defined DQM Regions and DQM Parameters up to the configurable level. The term ‘region’ here refers to any scope of data quality information within which there can be a hierarchy of evaluation. When a DQM Agent process is created it gets the list of *[DQM Parameter; Tree Depth]* pairs, which defines the scope of responsibility of this agent. For example consider the DQ tree shown on Fig. 3 organized around the SCT detector regions.

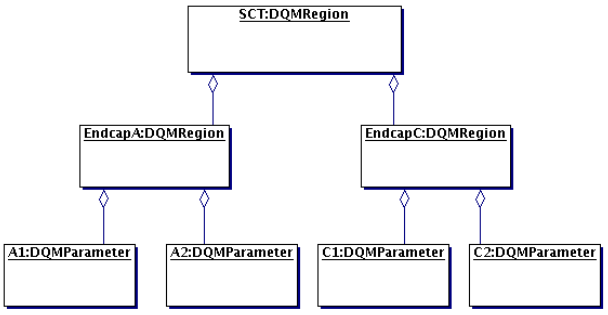

Fig. 3: An example of DQ Tree.

There is a number of ways in which responsibility can be distributed between different DQM Agent processes:

[1] One DQM Agent is started with the *[SCT;3]* parameter. It will be responsible for the whole DQ tree. Only one agent is necessary in this case.

[2] Three DMQ Agents are started with the *[DQRegion$_i$;2]* parameters, one for each DQMRegion.

[3] Seven DQM Agents are started with *[DQRegion$_{i,,}$ DQParameter$_i$;1]* parameters, one for each DQMParameter and DQMRegion.

The functionality described here is meant to be general in its accommodation of scalability. For instance, while there are clear hierarchies of organization of data quality results for sub-detector elements, other vantages are possible. One can consider data quality results that are organized around reconstructed objects like electrons, jets or Etmiss. In analogy to the scenario shown in Fig. 3 for SCT data quality, we can envision a Jets hierarchy under an overall Jets DQM Region. There might be Calorimeter, Tracking, and Calibration DQM Regions under this. The lowest level under Calorimeter might include Width and EM fraction DQM Regions. The users, whether they are detector, offline reconstruction, or trigger algorithm experts will determine the specific hierarchies actually chosen.

B. *Deployment*

The diagram on Fig. 4 shows a possible distribution of the DQMF processes on a number of nodes, which have to be connected via the ATLAS Control Network.

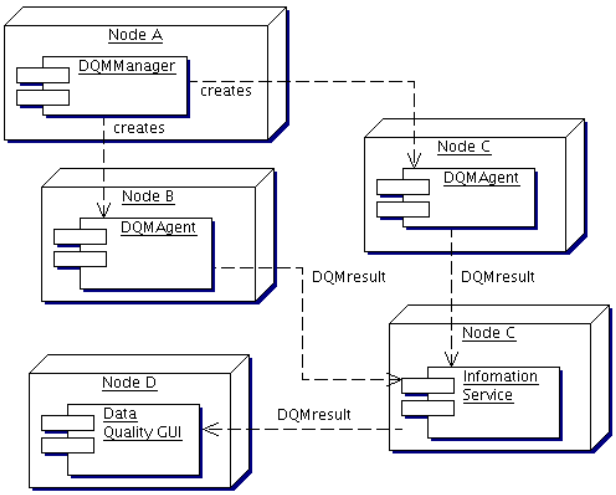

Fig. 4: Example of DQM processes distribution on five computing nodes.

This distribution is just an example that illustrates the approach of addressing the DQMF scalability issue. For small DAQ partitions, which don't require substantial processing power for the DQ analysis, various DQMF processes can be deliberately combined with each other to be executed on a smaller number of computers.

VI. Implementation

Different engines are implemented (Fig. 5) in order to be able to run the DQMF in online and offline environments or in algorithm development mode (ROOT).

A. *DQM Agent*

This application represents the online engine of the DQMF. DQMF may contain one or more DQM Agents with each of them responsible for a well defined subset of the whole DAQ system. Each DQM Agent evaluates the information that is used for the DQ assessment and produces the DQM Result for all the individual components of that subset as well as an overall DQ status for this subset as a whole.

B. *DQM Workbench*

The algorithms which are running in the framework can also be used standalone in ROOT, or the Workbench. In this environment developers get a chance to test the algorithms without setting up any configuration database and without any dependency on the framework. The aim of the workbench is to help determine the best configuration for a given DQParameter. The tuning of the DQParameter objects can be very sensitive and developers need this experience to understand the behavior once plugged in to the framework.

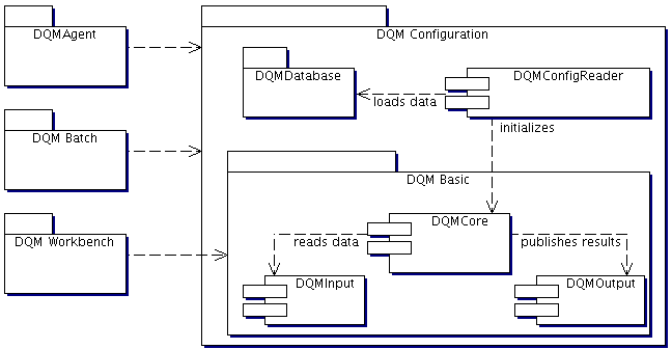

Fig. 5: Different environments to run DQMF.

C. *DQA offline*

The offline DQM framework, DQA, is similar to the online version in functionality. The data source, output, and configuration are the only differences. DQA uses a file based input and output. In order not to have additional dependency on online release from offline, a simple text based configuration file is used to configure DQA. This configuration file can then be bundled with a reference root histogram, if needed, into a root file object. In this manner the information needed to perform the algorithm is provided from one source. This functionality is needed in the tier-0 environment.

VII. Conclusions

The first implementation of DQM core has been available since Jan 2007. It includes 24 algorithms, a default summary maker, and a workbench for developing and testing algorithms.

The initial implementation of the framework was for online use, DQMF. It has been, and continues to be, used during ATLAS commissioning activities. The system also includes an online GUI for displaying results as they are made available by the framework.

A prototype of the offline implementation has also been provided recently. The aim is to use this implementation in tier-0 tests during this summer.

References

[1] Rene Brun & Fons Rademakers , "Root: An Object Oriented Data Analysis Framework," [Online]. Available: http://root.cern.ch/.

[2] ATLAS Collaboration, “ATLAS technical co-ordination technical design report,” CERN/LHCC/99–01.
[3] ATLAS Collaboration, “ATLAS detector and physics performance technical design report,” CERN/LHCC/99–14.
[4] ATLAS Collaboration, “ATLAS DAQ, EF, LVL2, and DCS”, CERN/LHCC/98-16.
[5] Kolos, S. *et al*, “Experience with CORBA Communication Middleware in the ATLAS DAQ, CERN-ATL-DAQ-2005-001.
[6] Conde-Muino, P. *et al*, “Portable Gathering System for Monitoring and Online Calibration at ATLAS**”,** CERN-ATL-DAQ-2004-014.
[7] Liko, D. *et al*, “Control in the ATLAS TDAQ System**”,** CERN-ATL-DAQ-2004-013.
[8] F. Zema, "Monitoring Data Archiving", Proceedings of IEEE NSS/MIC 2006 Nuclear Science Symposium, Medical Imaging Conference, October 29 - November 4, 2006, San Diego California.
[9] “ATLAS:ONASICUserGuide”, [Online]. Available: http://sim.fc.ul.pt/sim_en/ATLAS:ONASICUserGuide.
[10] R.Jones, L.Mapelli, Yu.Ryabov, I.Soloviev, “The OKS Persistent In-Memory Object Manager”, IEEE transaction on Nuclear Science, August 1988, Vol 45, No.4, pp.1958-1964 (ISSN 0018-9499).