The ALICE-LHC Online Data Quality Monitoring Framework: Present and Future

Filimon Roukoutakis, Sylvain Chapeland
CERN Physics Department
CH-1211, Geneva 23, Switzerland
Filimon.Roukoutakis@cern.ch

Özgür Çobanoğlu
INFN Turin
Turin, Italy

***Abstract*— ALICE is one of the experiments under installation at CERN Large Hadron Collider, dedicated to the study of Heavy-Ion Collisions. The final ALICE Data Acquisition system has been installed and is being used for the testing and commissioning of detectors. The Online Data Quality monitoring is an important part of the DAQ software framework (DATE). In this presentation we overview the implementation and usage experience of the interactive tool MOOD used for the commissioning period of ALICE and we present the architecture of the Automatic Data Quality Monitoring framework, a distributed application aimed to produce, collect, analyze, visualize and store monitoring data in a large, experiment wide scale.**

I. Introduction

A. The ALICE experiment

High Energy Experimental Physics has established and validated over the last decades the theory of fundamental particles and their interactions known as the Standard Model of Particle Physics. Heavy-Ion High Energy Physics aims to extend the Standard Model to complex and dynamically evolving systems of finite size. Of specific interest is the Physics of strongly interacting matter under extreme conditions of energy density.

ALICE (A Large Ion Collider Experiment) [1]–[3] is the LHC (Large Hadron Collider) experiment dedicated to the study of Heavy-Ion Collisions at CERN. It will focus on the study of collective strong interactions and quark-gluon plasma formation signatures. It consists of several detectors of different types and is designed to cope with very high particle multiplicities (dN_{ch}/dy up to 8000). Commissioning of detectors is progressing now in the underground experimental pit at the Swiss-French borders. The detectors along with the required support services are expected to be operational by LHC startup.

ALICE (Fig. 1) consists of a central cylindrical barrel embedded in the large L3 experimental magnet and a set of detectors installed on forward regions. The central part covers the polar angle range $[45^0, 135^0]$ over the full azimuth and consists of the following detectors starting from the innermost: Inner Tracking System (ITS) composed of six layers of different types of silicon detectors (pixels, strips and silicon drift chambers), Time Projection Chamber (TPC), Transition Radiation Detector (TRD) and Time Of Flight (TOF) which is based on multi-gap resistive-plate chambers. In the central part also resides the High Momentum Particle Identification Detector (HMPID) which uses Ring Imaging Cherenkov (RICH) technology, the PHOton Spectrometer (PHOS) and the ElectroMagnetic Calorimeter (EmCal). These last three detectors provide partial azimuthal coverage. In the forward region resides the Muon Arm which is composed of a dipole magnet, absorber and tracking-triggering stations and also several smaller detectors: The Photon Multiplicity Detector (PMD), the Zero Degree Calorimeter (ZDC), the Forward Multiplicity Detector (FMD), V0 and T0. Finally, on top of the L3 magnet resides ACCORDE serving as a cosmic ray trigger.

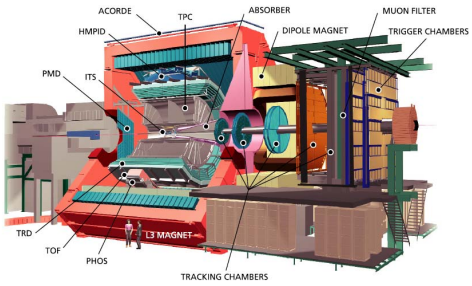

Fig. 1. ALICE Longitudinal view

ALICE will operate in several running modes with significantly different characteristics. The experiment has been designed primarily to run with heavy-ion beams, which are characterized by relatively low interaction rates ($\leq 10\ KHz$ for $L = 10^{27} cm^{-2} s^{-1}$), short running time (in the order of a few weeks per year) but very complicated event topology with high multiplicity and event size. For pp or pA running mode, the interaction rates are much higher (up to 200 KHz) but the event size is small and the running time is several months per year.

B. The ALICE Data Acquisition system

One of the functions of the DAQ system [4] (Fig. 2) is to realize and control the dataflow from the detector up to

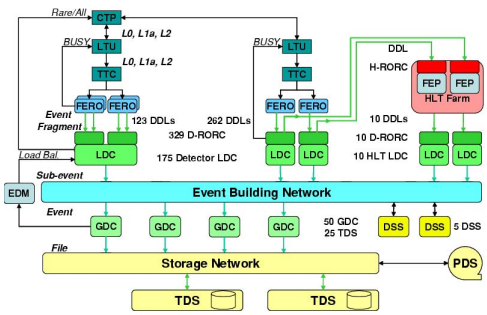

Fig. 2. ALICE DAQ Hardware Architecture

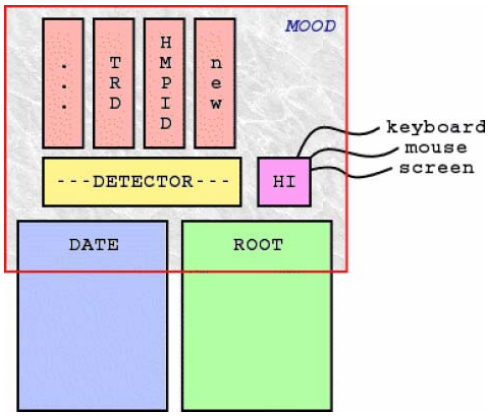

Fig. 3. MOOD plugable design

the data storage and perform the event building. The dataflow starts at the detector Front End Electronics (FEE) where the hardware and software interface to the rest of the DAQ system has been standardized via the Detector Data Link (DDL). DDL consists of a bi-directional optical link between the Source Interface Unit (SIU) and the Destination Interface Unit (DIU). The SIU is connected to the FEE while the DIU is interfaced to the D-RORC. D-RORC (DAQ Read-Out Receiver Card) is a PCI module hosted in commodity PCs called LDCs (Local Data Concentrators). The role of D-RORC is to inject raw data coming from DDL in the LDC memory where software is responsible for sub-event building of event fragments coming from all the DDLs connected to the same LDC. The D-RORC can also send configuration commands or data to the FEEs through the DDL. The LDCs dispatch the sub-events to a farm of PCs called GDCs (Global Data Collectors) that perform the final event building. The communication between LDCs and GDCs is achieved through the event building network via TCP/IP, capable to reach an aggregate bandwidth in the order of $2.5\ GB/s$.

DATE (Data Acquisition and Test Environment) [5], [6] is a software framework that has been developed to coherently drive the operation of ALICE DAQ. DATE is composed of packages that perform the different functionalities needed by the DAQ system. These include low level functionalities such as memory handling, process synchronization and interprocess communication and higher level functionalities like DDL readout, event building, data recording, runcontrol, information logging, error handling. DATE utilizes the DIM (Distributed Information Management) [14] system for interprocess communication between different nodes, the SMI++ (State Management Interface) [15] system for process control, MySQL database management system [17] for all the configuration databases and Tcl/Tk [18] libraries for GUI implementation. DATE also relies on standard Unix facilities like pipes and TCP/IP. The operating system of choice for the DAQ enviroment is Scientific Linux CERN 4 [19].

C. High Level Trigger

Related to the on-line environment is the High Level Trigger (HLT) of ALICE, a farm of several hundreds of PCs aimed to perform optimized reconstruction algorithms at data-taking time in order to take trigger decisions. Such a facility is also known in other experiments as Level 3 Trigger or Event Filter Farm.

D. Data Quality Monitoring

Data Quality Monitoring (DQM) is an important aspect of every High Energy Physics Experiment. Especially in the era of LHC where the detectors are extremely complicated devices and advanced on-line algorithms take real-time decisions that reduce the data volume up to 6 orders of magnitude, it is evident that a feedback on the quality of the data that are actually recorded for offline analysis is of extreme usefulness.

DATE provides a low-level monitoring package which forms the basis of any high-level monitoring framework for ALICE. It exposes a uniform API for accessing on-line raw data on LDC and GDC as well as data written in files. It gives the possibility of selecting the event sampling strategy for on-line streams in order to balance the needed computing resources.

II. Interactive Data Quality Monitoring: MOOD

MOOD (Monitor Of Online Data) [10], [11] is the project aimed to serve the interactive DQM needs of ALICE. It is written in C++ and makes heavy use of the ROOT framework [7]–[9]. ROOT is used to provide the GUI and the analysis tools such as histograms and graphs. The DATE monitoring library provides the needed interface to the DAQ. MOOD has plugable structure (Fig. 3). The executable is essentially a ROOT GUI application in which classes containing the detector specific functionalities are loaded at runtime. These functionalities include the desired visual layout and the detector specific analysis on the raw data. For a detailed description of the application capabilities in its initial form for the period 2002–2006 cf. [10]

Four years after the initial release and after several detectors had implemented MOOD modules for their laboratory tests it was decided, in view of the upcoming automatic and distributed DQM framework, to restructure MOOD in order to address the following needs:

- Experience has shown that many detector modules had common functionality repeated in the code which could be factorized in a backend that provided a simple and user-friendly API. This would minimize the impact of future changes in ROOT/DATE API and ease the maintenance of modules.
- Reviewing the detector module implementations it was possible to find common usage patterns and define a strict sequence of operations generic enough to allow all the usage cases already present. This would provide a partial preview of the distributed DQM framework.

The restructuring of the application was facilitated by the fact that it has a plugable structure. Keeping backwards compatibility for the existing detector modules was a key requirement. For this reason it was decided that only cosmetic changes would be made in the central GUI part and that a completely redesigned backend would be developed that was essentially only visible by newly implemented modules. Some additional design decisions have been:

- The interface to the DATE monitoring library should be isolated from the main application as a separate library.
- The framework should provide seamless access to AliRoot, the ALICE offline framework [12].
- Since several modules required custom decoding functionality not related to ROOT/AliRoot it was decided that this code should be isolated in a separate library, still in the same source distribution, that could in the future be provided as a completely separate package to be used for example by the upcoming DQM framework.
- Although interactive, the application should be able to provide a fast and efficient method for the analysis of large number of events in real-time, effectively simulating a batch analysis process in this respect.

The above decisions were implemented as follows:

The DATE++ library was created as an integral part of the MOOD distribution, encapsulating the DATE monitoring library API to a C++ class library and providing two additional functionalities:

- DATE raw event parsing: TDATEventParser class either parses the event and creates an index of the contained subevents/event fragments or retrieves the requested event fragment on demand.
- Efficient hexadecimal dump of the payload: TDATEEventDumper class provides very fast dumping of the payload in a variety of human readable forms. It is characteristic that -mainly due to the nature of console I/O-, dumping an event as a series of hexadecimal values displayed on a ROOT GUI window is faster than the reference implementation of DATE, eventDump console application.

Access to AliRoot is simply provided by changes in the Makefile of the application. However, a strong requirement was that MOOD should also build without AliRoot present. The solution to this problem was to enclose the AliRoot depending code into preprocessor conditional compilation statements. This practice must also be followed coherently by detector modules to ensure proper building of MOOD.

For the custom decoders a separate source tree was created. Building it produces a separate shared library. The only requirement for code to reside in this library is that it is pure C++ code without any external dependencies, including ROOT and AliRoot. Another less strong requirement is that decoders are event fragment level based, i.e. they can decode (and possibly map) a single DDL event fragment. As an initial effort a generic RCU and a generic DRM decoder have been implemented. RCU (Readout Control Unit) is the FEE readout card used for the readout of 4 ALICE detectors (TPC, FMD, PHOS, EmCal). DRM (DAQ Readout Module) is the FEE readout card utilized by TOF and T0. These decoders require an external mapping in the form of a file in order to associate the electronic channels with the payload words. The clear separation of the mapping from the main decoder facilitates the frequent changes of mapping that occur in test and commissioning conditions where usually only specific parts of the detector are tested. These decoders were subsequently used in combination with MOOD monitoring modules for TPC and T0. It is our goal that this very efficient Online Decoding Library for the ALICE experiment will serve as a positive contribution to the offline raw data related code.

To accomplish almost real-time operation it was decided to effectively separate the Physics analysis from the update of the screen. This is realized through a heuristic definition of the algorithm that drives the application functionality, described hereafter.

The above changes were complementary to the needed major redesign of the framework backend mentioned previously. We decided to implement the backend using the Object Oriented version of the "Template Method" behavioral design pattern [13]. This pattern is also referred to as the Hollywood Principle: "Don't call us, we will call you" and relies on the simple observation that if there exist a strict sequence of operations for an algorithm, this sequence can be implemented as a series of calls on member functions of an Abstract Base Class (ABC). Then, any user class derived from this ABC can redefine the pure virtual functions of the base class and essentially execute custom code at the time dictated by the predefined call sequence.

The described changes can be summarized in the simplified static class UML diagram of Fig. 4

TMMainFrame class serves as the "plugger" for the modules and provides the generic GUI functionalities (menu bar, status bar). TMBaseModule is the Abstract Base Class mentioned previously. The 6 pure virtual functions that essentially drive the monitoring process are:

1) InitMonitors: This is mainly used in case user code needs to acquire configuration/values settings from the GUI and store them in C++ variables so they are accessible by the algorithm at runtime. For example, if a user enters a value in a textbox this is the function where this value should be copied in a variable.
2) ResetMonitors: This function is called whenever there

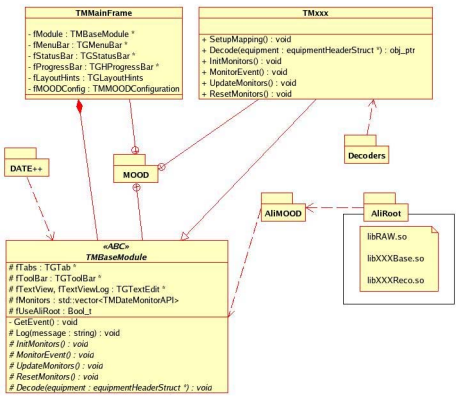

Fig. 4. MOOD static class UML diagram

is a need to reset the histograms, for example when the "Reset" button is pressed.

3) UpdateMonitors: This function is called to update the screen. User code selects the appropriate pad where a histogram should be drawn and issues a screen drawing command.
4) PreMonitor: The code in this function is executed after the update of the screen. Update of the screen can happen either after a single event monitoring or after a specified period of monitored events. It follows that in the second case analysis is done on all the events in the background.
5) PostMonitor: The code in this function is executed before the update of the screen.
6) MonitorEvent: This is the function where actual analysis takes place. The event fragments are accessible by the MOOD API and user code can subsequently decode the payload, perform analysis and store the results, usually by filling appropriate histograms. The MOOD API provides all the information for the event (event type, trigger masks) that are available by DATE. These information can be used to differentiate the behavior of MonitorEvent according to the event type.
7) In practice, although not enforced by the framework, it is helpful for each module to contain two additional functions for the histogram booking and the GUI creation. The usual name is ConstructMonitors and ConstructGUI respectively.

MOOD supports essentially two modes of operation (see also Fig. 5 for the buttons mentioned):

- By pressing "Get Event" button the following call sequence is implemented: InitMonitors, PreMonitor, MonitorEvent, PostMonitor, UpdateMonitors
- By filling in the appropriate textbox the values for the total number of events N to be monitored as well as the update period M of the screen and pressing the "Start Event Loop" button the following call sequence is implemented: InitMonitors, $N/M\times$ (PreMonitor, $M\times$ MonitorEvent, PostMonitor, UpdateMonitors)

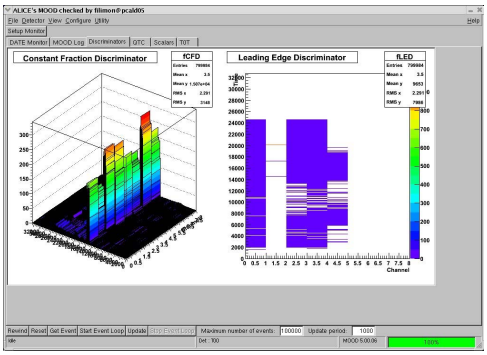

Fig. 5. Example of MOOD main screen view

MOOD now has a solid design. It has been and it is still used in several lab setups and test-beams with only minor non-design issues that are fixed as soon as possible. We expect that MOOD can and will have an important role during the commissioning phase and the first months of LHC running. Support for the project will continue as long as there is user demand for it.

III. Automatic Data Quality Monitoring: AMORE

The complexity of LHC experiments like ALICE imposes some fundamental requirements on the design of a modern Automatic DQM framework. Gaining from the experience of MOOD, some additional requirements were set:

- MOOD is a single process application capable of accessing a single monitoring source at a time. While one can argue that accessing a global GDC event could grant the ability to correlate data between detectors, this would create a network overhead for all monitoring programs which access only a small part of the data and it is doubtful whether the processing power of a single core would suffice for such advanced functionalities as fast reconstruction or correlation between different detectors. Automatic DQM requires a many-to-many client-server paradigm to serve the diverse monitoring needs.
- There is strong dependence on detector code being part of the MOOD source distribution. This requires collaborative development scheme through -for example- CVS (Concurrent Versioning System) [16] which is not convenient for several reasons. Instead it was decided that a scheme is devised where the offline CVS, already holding detector specific code, is used.
- The Automatic DQM will perform data analysis and issue alarms in case of problems. This functionality requires batch processes that are closely related and controlled by the DAQ or Experiment Control System facilities and not an interactive application.

The new framework is named AMORE (Automatic MOnitoRing Environment). The major design decisions taken to address the previous issues are:

- AMORE shall be a distributed application following the Observer design pattern, also known as publish-subscribe

paradigm, in which there exist a large number of information producers as well as a large number of information consumers and a many-to-many connection that is easily scalable can be established. This last requirement cannot be fullfilled with a simple server-client paradigm. The information producers have access to raw data from the DAQ network as well as published data of other producers. The information consumers can subscribe to producers, post-process and visualize the data.

- AMORE shall have no source dependence on detector code. This is essentially accomplished by heavy usage of C++ reflection. In computer science, reflection is the process by which a computer program of the appropriate type can be modified in the process of being executed, in a manner that depends on abstract features of its code and its runtime behavior. Figuratively speaking, it is then said that the program has the ability to"observe" and possibly to modify its own structure and behavior. The programming paradigm driven by reflection is called reflective programming. Typically, reflection refers to runtime or dynamic reflection, though some programming languages support compile time or static reflection. It is most common in high-level virtual machine programming languages like Smalltalk, and less common in lower-level programming languages like C. In C++/ROOT it can be accomplished by building source code dictionaries at compile-time.
- AMORE shall use the same interprocess communication and control mechanisms as the ALICE DAQ, namely DIM and SMI++.
- AMORE publishers and subscribers shall use intermediate pools for data exchange. The original implementation shall use MySQL servers for this purpose, although provision shall be made that the framework is not bound to a specific implementation.
- All configuration shall be handled via MySQL databases.

A development timeline for the distributed DQM application that would serve the needs of ALICE for the first years of running was established.

- May 2006–August 2006: MOOD reimplementation as a preview and technology testbed of the automatic DQM framework. No further architectural changes afterwards, only required bugfixes.
- September 2006–January 2007: Collection of Physics requirements for monitoring from detector groups, namely preliminary lists of histograms along with a short rationale for the Physics scope.
- January 2007–September 2007: Implementation of the new framework and test commissioning with selected detector groups, collection of additional framework requirements.
- October 2007–April 2008: Full detector integration, additional requirements implementation.

In figure 6 the UML collaboration diagram of the AMORE processes is presented. On the topmost level dqmAgents run

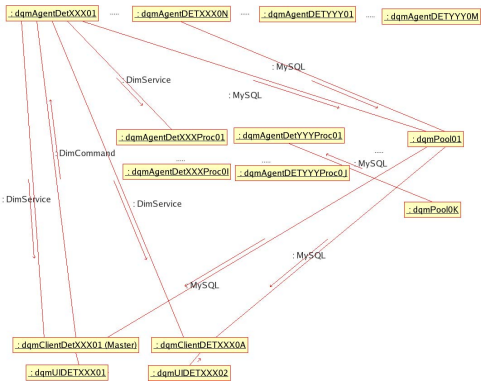

Fig. 6. AMORE UML Collaboration diagram

as batch processes. They have access to the level 2 trigger raw data from LDCs and GDCs. Their task is to decode and transform the raw data into physical quantities stored in the form of MonitorObjects. For the purpose of this analysis it suffices to consider MonitorObjects as histograms with additional housekeeping information that allow proper and coherent handling by the framework. dqmAgents have the ability to dispatch an instance of their MonitorObject content on a dqmPool. Each dqmAgent can connect to one and only one dqmPool for this purpose. In essence dqmAgents publish their results on dqmPools. On the opposite side, dqmClient processes subscribe to MonitorObjects on any dqmAgent. This allows dqmClients to receive regular updates on the content of the subscribed MonitorObjects. As it can be seen from the diagram, there is no direct MonitorObject transfer between dqmAgents and dqmClients. All such transactions occur via the dqmPools which are implemented as MySQL servers. The framework also provides the possibility of non-first level dqmAgents that only have access to MonitorObjects published by other dqmAgents. They can perform post-processing and publish their results.

Communication between any processes for the purpose of command or information exchange is done through DIM. Any batch process in the framework acts as a DIM server which can receive commands from any DIM client implementation. In practice, the concept of a master client per subsystem will be introduced. Only the master client will be granted privileges to send commands to the dqmAgents of the subsystem. The rest of the clients for this subsystem will be simple observers, only able to subscribe to desired MonitorObjects.

dqmAgents are implemented as finite state machines. This will facilitate the future integration with DAQ/ECS via the SMI++ framework. dqmClients are in reality modularized. There exist the actual dqmClient backend which is responsible for the subscription handling and contains an abstract interface for any client application and a higher level layer which is responsible for the visualization and utilizes the abstract interface of the dqmClient.

IV. Conclusion

We presented the architecture of the ALICE Interactive and Automatic DQM framework which has taken into account the existing experience from other LHC experiments. The architecture is generic enough but still encapsulates all the required elements needed for ALICE. The biggest advantages are the scalability and modularity of the framework. At the moment of this writeup implementation of the framework is ongoing. A fully functional prototype will be in place at LHC startup.

Acknowledgment

The authors would like to thank the ALICE DAQ group for providing the working environment that allowed the realization of the DQM concepts into actual code.

References

[1] ALICE Collaboration, 1995, *ALICE Technical Proposal for A Large Ion Collider Experiment at the CERN LHC*, CERN/LHCC/95-71, LHCC/P3, ISBN 92-9083-077-8

[2] ALICE Collaboration: F Carminati et al, 2004 *ALICE: Physics Performance Report, Volume I*, J. Phys. G: Nucl. Part. Phys. 30 1517-1763 doi:10.1088/0954-3899/30/11/001

[3] ALICE Collaboration et al, 2006 *ALICE: Physics Performance Report, Volume II*, J. Phys. G: Nucl. Part. Phys. 32 1295-2040 doi:10.1088/0954-3899/32/10/001

[4] ALICE Collaboration, 2004, *ALICE Technical Design Report of Trigger, Data Acquisition, High Level Trigger and Control System*, CERN-LHC-2003-062, ALICE TDR 10, ISBN 92-9083-217-7

[5] http://ph-dep-aid.web.cern.ch/ph-dep-aid/, *ALICE DAQ Project Web Site*

[6] ALICE DAQ Project, 2006, *ALICE DAQ and ECS User's Guide*, ALICE-INT-2005-015, CERN

[7] Brun R., Rademakers F. et al., 1997, *ROOT - An Object Oriented Data Analysis Framework*, Nuclear Instruments and Methods in Physics Research, A 389, 81-86

[8] Brun R., Rademakers F., Panacek S., Buskulic D., Adamczewski J., Hemberger M., West N. et al., 2006, *ROOT Users Guide 5.14*, CERN

[9] http://root.cern.ch, *ROOT Web Site*

[10] Cobanoglu, O.; Vande Vyvre, P.; Ozok, F., *Development of An On-Line Data Quality Monitor For The Relativistic Heavy-Ion Experiment ALICE*, Real Time Conference, 2005. 14th IEEE-NPSS, Stockholm

[11] http://ph-dep-aid.web.cern.ch/ph-dep-aid/MOOD, *MOOD Web Site*

[12] http://aliceinfo.cern.ch/Offline, *ALICE Offline Project Web Site*

[13] Gamma E. et al, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional Computing Series, ISBN 0-201-63361-2

[14] Gaspar C. et al, *DIM, a Portable, Light Weight Package for Information Publishing, Data Transfer and Inter-process Communication*, International Conference on Computing in High Energy and Nuclear Physics (Padova, Italy, 1-11 February 2000)

[15] Gaspar C. et al, *SMI++ Object Oriented framework for designing Distributed Control Systems*, Xth IEEE Real Time Conference 97 (Beaune, France, Sep 22-26 1997)

[16] http://savannah.nongnu.org/projects/cvs/ *CVS Web Site*

[17] http://mysql.org *MySQL Web Site*

[18] http://www.tcl.tk/ *Tcl/Tk Web Site*

[19] http://linux.web.cern.ch/linux/ *Scientific Linux CERN Web Site*