The Resource Manager the ATLAS Trigger and Data Acquisition System

I Aleksandrov 1, G Avolio 2, G Lehmann Miotto 2, I Soloviev 3

1 Joint Institute for Nuclear Research, Joliot-Curie 6, Dubna, Moscow region, Russia
2 European Organization for Nuclear Research, CH-1211 Geneva 23, Switzerland
3 University of California, Irvine, CA 92697, 949-824-5011, USA

Igor.Aleksandrov@cern.ch

Abstract. The Resource Manager is one of the core components of the Data Acquisition system of the ATLAS experiment at the LHC. The Resource Manager marshals the right for applications to access resources which may exist in multiple but limited copies in order to avoid conflicts due to program faults or operator errors. The access to resources is managed in a manner similar to what a lock manager would do in other software systems. All the available resources and their association to software processes are described in the Data Acquisition configuration database. The Resource Manager is queried about the availability of resources every time an application needs to be started. The Resource Manager's design is based on a client-server model, hence it consists of two components: the Resource Manager "server" application and the "client" shared library. The Resource Manager server implements all the needed functionalities, while the Resource Manager client library provides remote access to the "server" (i.e., to allocate and free resources, to query about the status of resources). During the LHC's Long Shutdown period, the Resource Manager's requirements have been reviewed at the light of the experience gained during the LHC's Run 1. As a consequence, the Resource Manager has undergone a full re-design and re-implementation cycle with the result of a reduction of the code base by 40% with respect to the previous implementation. This contribution will focus on the way the design and the implementation of the Resource Manager could leverage the new features available in the C++11 standard, and how the introduction of external libraries (like Boost multi-container) led to a more maintainable system. Additionally, particular attention will be given to the technical solutions adopted to ensure the Resource Manager could effort the typical requests rates of the Data Acquisition system, which is about 30000 requests in a time window of few seconds coming from more than 1000 clients.

1. Introduction

The Trigger and Data Acquisition [1] (TDAQ) system of the ATLAS [2] detector at the Large Hadron Collider (LHC) at CERN is composed of a large number of distributed hardware and software components (about 3000 machines and more than 25000 concurrent processes at the end of LHC’s Run 1) which provide the data-taking functionality of the overall system. The Resource Manager (RM) is one of the core components of the ATLAS online Data Acquisition system. The Resource Manager marshals the right for applications to access resources which may exist in multiple but limited copies in order to avoid conflicts due to program faults or operator errors. The access to resources is managed in a manner similar to what a lock manager would do in other software
systems. The Resource Manager is queried about the availability of resources every time any application needs to be started.

![Resource Manager Graphical User Interface](image)

**Figure 1.** Resource Manager Graphical User Interface

During routine ATLAS operations many applications must be started and stopped within a small time window. The Resource Manager is designed to handle of order 30k requests within a few seconds from O(1k) clients in the data acquisition system via a custom API. A GUI application is also available for use by experts to view and update resources as needed. The content of the Resource manager viewer is shown in figure 1.

2. **Resource Manager architecture**

The overall architecture of the RM is presented in figure 2. The RM component is essentially divided into RM server and client parts. The RM client performs requests to the RM server using CORBA based Communication [3]. The RM client uses access management [4] for some types of action before issuing requests in order to check if such actions are permitted for a given user. Clients can ask for allocation or release of resources, get information about allocated resources and update configuration data that is stored in the RM server dynamic database (RM DDB). RM clients use a configuration database [5] in order to load objects that correspond to resources and their associations.

![The Resource Manager Architecture](image)

**Figure 2.** The Resource Manager Architecture

The RM server runs as a single instance covering the entire TDAQ system. The RM server is a passive component and reacts only to client requests. It consists of an RM DDB and wrapper, which provides multi-threaded processing of DB requests as well as support for backups. The RM DDB keeps all data concerning allocated resources and corresponding allocation parameters like client, application, program identifier etc.

The RM server is essential to be able to control usage of resources while the TDAQ system is running. The server should therefore be robust and handle all failures in a graceful way including restarts in case of a crash and provide recovery of data from a backup. Data backups consist of a base file storing...
all RM DDB information and a journaling file storing any changes on top of the base file data. The base file itself is updated periodically. The journaling file is immediately updated when a corresponding action is completed successfully. When the RM server is started and restored from backup the base file is used to recover the bulk of the data and then the remaining actions are recovered from the journaling file by parsing records.

The RM server is implemented in C++. The client is implemented in C++ and in Java. Java is used mostly in different Java-based GUI applications calls. A Python implementation of the RM client functionality was recently developed for monitoring the states of different resources.

### 3. Resource Manager context

All the available resources that are used by the RM and their associations to software processes are described in the configuration database. The view of the Configuration DB classes related to RM resources is shown in figure 3. The RM uses the software object (SW_Object), RM resource (RM_Resource) and Computer in order to control RM resources. There are two types of RM resources in the ATLAS TDAQ configuration database. An RM Software Resource is used when only limited copies of some software may run in the ATLAS TDAQ system on any host. An RM Hardware Resource is used when only one application can start on the same host. RM Hardware Resources are associated with a software object which has no association with a computer. This greatly decreases the number of resources that should be stored in the database. The RM creates whatever hardware resources are needed on the fly when an application starts.

![Figure 3. Schema of Configuration DB classes related to RM resources.](image-url)
During run-time, any TDAQ application starts only if all associated resources are available. The RM server checks if the SW_Object of an application has associated RM resources and gives permission to start if no associations exist. If associations exist, the RM server checks if each resource associated with software object in global and partition scope is available. The RM server gives permission to run only if all associated resources are available. The RM resource is considered available if the number of allocated resources is less than the permitted limit (MaxCopyPerPartition and MaxCopyTotal in figure 3).

Most ATLAS TDAQ processes are started using the Process Manager [6] (PMG). The PMG is the main user of RM client functionality, sending requests to the RM server via the client API to allocate resources for applications to be started. Once a task is complete requests are also sent to the RM server to free any allocated resources. The PMG is designed to avoid situations where applications consuming resources crash and therefore do not free resources that they have allocated.

4. Resource Manager use cases
An example of the use of RM software resources is the Integrated Graphical User Interface (IGUI). The IGUI can run in control or in display mode as you can see in the figure 4 marked by the red ellipse. Only one copy of the IGUI can run with control rights and a limited number of IGUI instances can run in display mode. Two corresponding resources are defined in the configuration database. The IGUI uses the RM client java interface to request resources.

![Figure 4](image-url)

**Figure 4.** The IGUI view.

RM Hardware resources are most widely used in the ATLAS Readout System (ROS). The read-out cards (RobinNP) cannot be shared, meaning that only one RobinNP controller can start on the host PC. Just one hardware resource is stored in the configuration database. The PMG requests resources for the controller before starting it on a host using the host identifier as one of the request parameters. The RM server checks if the corresponding resource is already allocated and gives permission to proceed if it is free.

5. Resource Manager upgrades
During the LHC's Long Shutdown period, the Resource Manager's requirements were reviewed in light of the experience gained during the LHC's Run 1. Therefore, the Resource Manager underwent a full re-design and re-implementation cycle with the result of a reduction of the code base by 40% with respect to the previous implementation, leading to a more maintainable component. In particular, the server side code was greatly simplified and made modular in order to ease possible future upgrades, mainly in terms of functionality.
Figure 5. The plot of the Controller transition times.

As an example, some previously supported resource types requiring intensive computation and bookkeeping (like the amount of memory and CPU requested by processes on every single host of the system), were dismissed. Indeed the DAQ system evolved with time towards a process-scheduling schema allowing resource overbooking, making that kind of resource obsolete. The removal of complex and no longer used functionality made it possible to save more than 1600 lines of code including tens of different data structures and classes. Additionally, the usage of the new features available in the C++11 standard and Boost libraries [7] made it possible to re-organize the server’s internal data structures in a more efficient manner. For instance, it was possible to merge almost all the data structures used in the previous RM implementation in a single Boost multi-index container. The single container greatly simplified both the complexity of keeping the server internal data consistent and the query system used to extract information.

In a similar way, the introduction of the Boost archiving library helped to simplify the backup facility, which could effectively be implemented in few methods with 2-3 lines each of code.
Results shown in figure 5 give a hint of the raw performance the current RM system is capable of. The plot reports the time needed by a set of applications to execute an empty action (blue) or to ask the RM server for 10 Hardware resources (red). The blue data represent the overhead introduced to distribute commands to the applications and gather the results. The tests were executed on 48 nodes and different numbers of applications (1 to 30) started on a single node, giving a total of 1440 applications for the 30 applications per node case. As it can be noticed, the RM server managed to reply to 14400 requests in about 0.7 s, resulting in a request rate of about 20 kHz.

The RM should also have minimal influence on application start time. In order to prove that, a series of tests similar to the ones previously described were performed (see figure 6) on 48 nodes, with different numbers of applications started on a single node. One RM Hardware resource per application was requested in the usual way by the PMG. Figure 6 shows the average application start time for 2 cases, where requests to the RM were enabled or disabled. This demonstrates that there is no visible overhead on the system performance after enabling the RM.

The redesign of the RM Server did not lead to changes in the RM Client. The only additions to the client were some python functions to facilitate more easy access to resource information.

6. Conclusions
The Resource Manager Server runs as a single process and robustly controls all RM resources across the TDAQ system whenever applications are running.

New features available in the C++11 standard and Boost libraries helped to significantly simplify RM server re-implementation. Resource Manager server re-design and re-implementation made the component more maintainable, which is important for long-term support.
References
[1] The ATLAS Collaboration 2002 ATLAS high-level trigger, data-acquisition and controls: Technical Design Report.
[2] The ATLAS Collaboration 2008 The ATLAS Experiment at the CERN Large Hadron Collider, J. Instrum.
[3] Common Object Request Broker Architecture home www.corba.org
[4] Leahu M C, Dobson M and Avolio G 2008 Access Control Design and Implementations in the ATLAS Experiment IEEE Transactions on Nuclear Science 55, issue 1, pp. 386-91
[5] Lehmann Miotto G et al 2010 Configuration and control of the ATLAS trigger and data acquisition Nuclear Instruments and Methods in Physics Research A, 623, Issue 1, p. 549-51., 11
[6] Avolio G, Dobson M, Lehmann Miotto G and Wiesmann M 2008 The Process Manager in the ATLAS DAQ System IEEE Transactions on Nuclear Science 55, issue 1, pp. 399-404
[7] Boost C++ Libraries www.boost.org