A revised design and implementation of the ATLAS Log Service package

Raul Murillo Garcia\textsuperscript{1} and Giovanna Lehmann Miotto\textsuperscript{2} for the ATLAS TDAQ collaboration

\textsuperscript{1} Department of Physics and Astronomy, University of California, Irvine, California, USA
\textsuperscript{2} European Laboratory for Particle Physics (CERN), Geneva, Switzerland

Abstract. This paper presents a revised design and implementation of the Log Service for the ATLAS Trigger and Data Acquisition (TDAQ) framework at CERN. A previous version of this utility was rarely used for various reasons, herein explained. The lessons learned set the grounds and motivation for a new redesign. The Log Service consists of the Logger, the entity that collects logs and stores them in an Oracle database; a set of user utilities to access and maintain the database; and a Java based tool, known as the Log Manager, which provides a compact and intuitive interface for browsing the log messages based on a user defined search criteria. The outline of these software components are explained, including various optimization techniques deployed in order to handle the large volume of entries expected to be stored in the database. Finally, a performance study has been conducted to prove the validity and behavior of the Log Service.

1. Introduction

ATLAS (A Toroidal LHC ApparatuS) \cite{atlas} is aimed at studying high energy particle interactions at the Large Hadron Collider (LHC). The ATLAS Trigger and Data Acquisition (TDAQ) is a large computing system currently including 1,500 interconnected computers serving up to 15,000 applications. The TDAQ system orchestrates the readout, assembling, selection and storing of interesting collision data generated within the detector. Messages containing information issued by applications that form the TDAQ infrastructure are stored for system analysis and debugging. Historically, these messages are redirected to a log file with a unique name – a combination of the application name, the user name and the time the application was launched. To identify the log file, developers have to know where, who and when the application was launched. This solution is not ideal, as browsing logs among thousands of hosts, each containing dozens if not hundreds of files is rather cumbersome. In addition, log files are periodically archived and removed from the computers, making access to them even more difficult.

To address this issue, a logging mechanism was deemed essential early in 2005. Initially, Log4C++ was considered as a potential solution. However, in the view that the TDAQ infrastructure already provided tools for error reporting and data distribution, which were being standardized in all TDAQ applications, it was decided to develop an in-house Log Service package. Strictly speaking, the Log Service package described herein only includes the backend of the overall logging mechanism of the TDAQ system. As explained in the following chapter, this mechanism also includes the Error Reporting Service (ERS) and the Message Reporting System (MRS) \cite{ers_mrs}.
2. Error reporting in the TDAQ

The ATLAS experiment is operated in the context of partitions, described in configuration files. Each partition has a unique name and may include as little as few applications (for testing purposes) and as much as a full description of the ATLAS detector and the TDAQ infrastructure (for data taking purposes). A run is associated to a unique partition and encompasses a period of data taking with a defined set of stable conditions related to the quality of physics. Each run is identified with a unique run number. TDAQ applications are launched within the context of a partition and a run.

Message logging is done via the ERS, which offers a unified system for handling error, warning and debug messages, thus facilitating the logging frontend for the TDAQ. The ERS provides the interface to hook messages to different streams, like the standard output and error. One such stream is the MRS, a facility built on top of the Common Object Request Broker Architecture (CORBA). The MRS is used to report information messages to other components following the producer-server-consumer scheme. Figure 1 depicts an initial deployment diagram of the Log Service, where the arrow direction specifies the message flow. The left most node encompasses the hosts and applications that form the TDAQ infrastructure. Messages are issued via the ERS interface and transported by the MRS, which in turn, forwards them to the Logger, the component that collects and inserts all log messages in the Log Service relational database. Users can browse the logs either with a dedicated graphical user interface, the Log Manager, or via a set of command line utilities.

Figure 1. Diagram depicting the Log Service within the TDAQ system.

3. Initial implementation

The initial version of the Log Service [3], developed around mid 2005, employed MySQL as the relational database. The Logger and utilities were written in C++, whilst the Log Manager was a combination of PHP, HTML and JavaScript. This implementation did not catch on among developers and users due to several reasons:

- The Log Manager was not user-friendly. Messages could take minutes to be retrieved and displayed due to a poor database schema design. When the Log Manager was launched, the user was presented with a form-like WEB page where a search criterion had to be entered to narrow down the selection of retrieved messages. A second page would then display the log messages. This meant going back and forth to change the search criterion. The Log Manager had no configuration capabilities. Long messages were not easy to read as they were presented in a single table row.
- To make the Log Service scalable, several Loggers could be deployed, each handling a set of messages based on a severity level (FATAL, ERROR, WARNING, etc.). The number of Loggers and their associated severity level was kept in a unique table in a so-called Master Log Server database. This configuration had to be set up manually and was not intuitive.
- The code was highly dependent on the database technology deployed, yielding a very monolithic design.

---

3 The MRS service is a distributed system with a central node, the MRS server, and several so-called MRS workers. The former manages the client-server subscriptions by assigning a unique MRS worker to the producer-client channel. The MRS workers do the actual forwarding of messages to the MRS receivers.
4. Revised design and implementation

Two reasons triggered the redevelopment of the Log Service in mid 2007. Firstly, as explained before, the Log Manager and the command line utilities were seldom used. Secondly, a new ATLAS-wide requirement was introduced to use Oracle as the default relational database technology. Based on the lessons learned, the new Log Service implementation was design around the following requirements:

- The design had to set the basis for a properly structured code, thus yielding a flexible and modular package. There had to be a neat decoupling between the software and the deployed underlying relational database technology.
- The Log Manager had to be configurable, compact and user-friendly.
- Access to the database had to be reasonably fast (less than a minute).

4.1. Database schema

Figure 2 shows the Log Service database schema, consisting of three tables. Logs are inserted in the table LOG_MESSAGES, which contains a set of columns that map the fields in a log message - plus the partition and the run number in which the log has been issued. The number of entries in this table is expected to be of the order of millions. To minimize the retrieval time, the schema deploys the segmentation feature of Oracle databases. This enables a table to be segmented for easier management and improved performance. To make this possible, the partition name (string) is associated to a unique partition ID (integer). This mapping is kept in the table PARTITION_IDS. Inserting a new entry in this table fires an Oracle trigger, which creates a new Oracle segment in the LOG_MESSAGES table (relevant to the new partition ID). The efficiency resides in the Oracle engine not having to traverse the whole LOG_MESSAGES table, but only the segment or sub-table associated to the partition ID. The choice of the partition as the Oracle segment key is a natural one since this parameter is at the top of the log message grouping hierarchy. Indeed, users must always define at least the partition name they want to search from - there is no use in browsing all partitions.

Further optimization has been achieved by indexing the columns USER_NAME and RUN_NUMBER in the table LOG_MESSAGES. Indexes are structures associated with tables that allow SQL statements to execute more quickly. This schema implies that efficient queries are only possible if the triad partition name, user name and run number are defined.

Finally, the METADATA table records information for each run: the partition ID, the user who ran the partition and the unique run number associated to it. The usefulness of this table will become more relevant when explaining the Log Manager (Section 4.3) and is related to the enforcement to choose the triad mentioned above.

4.2. Log Service design

Figure 3 shows the internal components and binaries of the Log Service and the interaction with external packages. The Database Interface refers to a collection of C++ classes that provide a homogeneous set of methods for reading from, writing to, creating and dropping the tables that form the Log Service database schema (see Figure 2). These classes make use of the COmmon Relational Abstraction Layer (CORAL), a CERN in-house package that provides functionality for accessing data in relational databases using a C++ and SQL-free interface. The Logger and Utilities components use the Database Interface and provide a well defined functionality for the TDAQ infrastructure, allowing users to retrieve and remove messages based on a search criterion (entered as command line...
arguments) as well as to get a list of the partition names with entries in the database. Note how the Logger uses the MRSReceiver interface to collect the log messages transported by this service.

The design of the Database Interface is based on layers, each adding a new level of abstraction from the database technology to the application level. Figure 4 presents this abstraction with each layer horizontally aligned. The bottom layer groups a set of classes that wrap around CORAL and provide the logical elements for a database connection, namely, a Table, a Session, a Transaction and a OneRow. The underlying technology is thus shielded from the user, an important requirement of the Log Service redesign. The next layer deals with actual representations of tables and rows that constitute the Log Service database schema (as described in Section 4.1): LogMsgsTable, PartitionIDTable and LogMetadataTable for tables and LogMessage, PartitionIDRow and MetadataRow for their associated rows. These classes are instantiated and their interface used by the MetadataWriter and the Logger. The former inserts a new entry in the METADATA table when a new run is started. The latter receives all the messages issued by the TDAQ applications and inserts them into the LOG_MESSAGES table. If the Logger is launched within a new partition, it records its name and its associated unique ID in the PARTITION_ID table. As explained before, this operation triggers the creation of a new Oracle database segment. The Logger component is thus an instantiation of the Logger class. From the point of view of the TDAQ infrastructure, the Logger is an MRS client – it subscribes itself to the MRS server. Finally, the top layer groups a collection of miscellaneous classes that provide a unique functionality for the MetadataWriter and the Logger classes. Thus, LogBuffer is a container for log messages providing the canonical operations insert(), remove(), size(), etc. The Logger queues the incoming messages into this temporarily buffer until it reaches a user-definable threshold. When this occurs, the logs are flushed into the LOG_MESSAGES table via a bulk insert. This optimizes the access to the relational database making the transaction more efficient. To guarantee that messages are not indefinitely sitting in the buffer until the threshold is reached, a message collector is scheduled to run periodically in order to flush the buffer. The LogTriggerCondition class serves as a synchronization point for the different threads to acknowledge the need to flush the log buffer. The classes LogServiceStats and LogServiceInfo are used to gather and publish statistics.

![Figure 3. Internal and external components.](image)

![Figure 4. Class diagram design.](image)

### 4.3. Log Manager

The Log Manager is the user graphical interface to browse the log messages produced by the TDAQ system. This tool has been developed in Java due to its flexible and powerful graphical libraries (AWT and Swing) and the well documented Java Database Connectivity library (JDBC), an industry standard for database-independent connectivity. Furthermore, Java based applications can take advantage of the Java Web Start Technology to robustly be deployed to the user desktop using a browser-independent architecture. The Java Web Start software downloads all the needed files for the application, as well as the Java Runtime Environment (JRE) if the requested version is not available locally. Giving access to the log messages outside the CERN boundaries was crucial due to the collaborative nature of the ATLAS experiment, which groups around 2,500 physicists and engineers across 37 countries.
One of the primary goals when redesigning the Log Manager was to achieve an intuitive and compact interface. This advocated for a tool with a single window containing six distinctive areas, as shown in Figure 5.

![Image](image.png)

**Figure 5. Snapshot depicting the Log Manager interface.**

The partition tree panel, on the left hand side, groups the log messages in a hierarchical structure, having the partition and user name as top and middle nodes, and the run number as the leaf objects. This tree is efficiently built by traversing the METADATA table, which is created for this very purpose. This interface enforces the selection of a partition, thus exploiting the Oracle segmentation feature, and further encourages the selection of a user name and a run number to fully exploit the optimization mechanisms deployed in the database schema.

The actual messages are presented in the log message table. Two search criteria panels are available: one associated to the partition tree, and the second associated to the log message table. The former is aimed at facilitating the search of a given partition, user name or run number, whilst the latter allows narrowing down the list of messages retrieved. At the bottom of the interface, the content detail panel displays the messages in a detailed manner. This is important to guarantee user-friendliness since long messages may be difficult to read in the log message table. This panel also outputs errors, warnings and information. To allow configurability, a pull down menu is available with multiple options.

The design of the Log Manager revolves around the six parts of the interface described above. Each part is coded as a unique class, which inherits from javax.swing.JPanel and configures the look and feel of the panel as well as the actions to be taken upon user interaction. The Java entry point class is where the interface is created by instantiating all the graphical panels and elements. Access to the database is transparently hidden by using the JDBC library thus decoupling the underlying relational database. Each time the user triggers the retrieval of messages, an independent thread is created to handle the specific SQL query. This prevents the Event Dispatch Thread – and thus the Log Manager – from being blocked, thus allowing users to perform other selections without waiting for the previous one to complete. Care has been taken to make sure that the messages presented in the log message table are those associated to the last selection.

5. **Performance study**

Due to length restrictions of this paper, only a brief summary of the performance study can be reported herein. For this study, a controllable framework has been set up involving a purposely created partition that instantiates across thirty hosts log producers, a utility that generates logs at a user defined rate. The metrics recorded in the Logger include the number of messages received, the number of messages buffered and the number of messages discarded. Messages are discarded when the Logger’s buffer overflows - the default size is 2,500 logs.
With this set up the rate of messages produced was increased up to roughly 6,000 msgs/sec. With a production rate of O(100 msg/s/sec) the Logger’s buffer remained very low and could easily be flushed. In Figure 6 the period between log transmissions is 1 ms. and the instances of log producers launched on each host is increased from 1 to 10. This configuration yields an average rate of 850 and 3,738 messages per second, respectively. The Logger can sustain both rates since no messages are reported as discarded and the buffer occupancy tops 1,000 messages. It is only at the end of the test, when there is a peak of around 5,500 msgs/sec that the buffer overflows and the Logger has to discard incoming messages. This did not cause the Logger to malfunction.

To contrast this study with the full-fledged deployment of the ATLAS experiment, the messages produced during the first week of beam in the LHC (week 48 of 2009) were analyzed. Right before the LHC start, the messages sent by the applications in the TDAQ framework averaged 2,000 msgs/sec. However, during collisions the rate decreased drastically to just a few logs per second, which has also been observed during the third quarter of 2010. This is expected since during normal ATLAS operation all the subsystems must behave correctly and logs account only for status information. Thus this implementation of the Log Service is sufficient to handle the log production in the TDAQ.

6. Conclusion
Redeveloping the Log Service has proved the benefits of refactoring. The development time was much less than that of the original version due to the existing knowledge and the better understanding of the requirements, which lead to a quick design and implementation. Refactoring is not always viable, especially for very large projects, constrained deadlines or scarce man power. Fortunately, the Log Service is a relatively small and contained software package that could be redeveloped well ahead of the ATLAS switch on. This has yielded a design that is better layered and a complete decoupling of the database technology deployed through the use of CORAL and JDBC.

Currently, the new version of the Log Service, deployed during the first term of 2008, is considered a mature application within the TDAQ system. Although the development is complete, occasionally, new features are added on user request, which is seen as a positive sign of the level of usage and the user willingness to improve this software package. Indeed, the Log Manager is now widely used and has been established as an essential tool for shifters in the ATLAS control room.

7. References
[1] ATLAS Collaboration, The ATLAS Experiment at the LHC, 437 pp., JINST 3:S08003, 2008.
[2] I. Fedorko et al, The Message Reporting System of the ATLAS DAQ System, ATL-DAQ-CONF-2007-029, presented at 10th ICATPP Conference on Astroparticle, Particle, Space Physics, Detectors and Medical Physics Applications, Como, Italy, 8-12 Oct 2007
[3] R. Murillo Garcia, G. Lehmann Miotto, A log service package for the ATLAS TDAQ/DCS group, Nuclear Science, IEEE Transactions on V. 54, Issue 1, Feb. 2007 Page(s):202-207, 2007