Usage of Message Queueing Technologies in the ATLAS Distributed Data Management System

Philippe Calfayan\textsuperscript{1}, Zang Dongsong\textsuperscript{2}, Vincent Garonne\textsuperscript{3}

\textsuperscript{1}Ludwig-Maximilians-Universität, Munich, Germany
\textsuperscript{2}Institute of High Energy Physics, Beijing, China
\textsuperscript{3}CERN, Geneva, Switzerland

E-mail: philippe.calfayan@physik.uni-muenchen.de

Abstract. The ATLAS Distributed Data Management system is composed of semi-autonomous, heterogeneous, and independently designed subsystems. To achieve successful operation of such a system, the activities of the agents controlling the subsystems have to be coordinated. In addition, external applications can require to synchronize on events relative to data availability. A common way to proceed is to implement polling strategies within the distributed components, which leads to an increase of the load in the overall system. We describe an alternative based on notifications using standard message queueing. The application of this technology in the distributed system has been exercised.

1. Introduction

The ATLAS Distributed Data Management (DDM [1]) system includes several services and external tools that are employed to store and access the data recorded or simulated at the ATLAS experiment. A huge amount of ATLAS data (larger than 30 PB) are registered in datasets within a central dedicated Oracle database. The operation of the DDM requires the activity of several agents that multiply requests to this database, and thus increase the load in the central services. Notifications via message queueing have been tested as an alternative technology to improve the overall DDM system.

2. Message queueing (MQ) with ActiveMQ

ActiveMQ [2] is a message broker under Apache 2.0 license. It supports multiple client protocols and allows various languages, such as Java, C, C++, C#, Ruby, Perl, Python, or PHP. In our study, we utilize the STOMP protocol [3] (Streaming Text Orientated Messaging Protocol) to interface with ActiveMQ in Python, while TCP is used to manage transactions.

Notifications processed by ActiveMQ can be handled mainly in two ways with respect to subscribers. We distinguish:

- Queues, for which any subscriber who consumes a message removes it from the queue
- Topics, for which multiple subscribers can consume messages independently

ActiveMQ supports persistence with durable subscriptions to queues or topics, which means that messages received while subscriber is disconnected are seen by the subscriber on next connection (a client identification is associated to the subscriptions). Furthermore, an expiration
time can be attributed to the messages, which can also be consumed in sizeable bulks by setting up a prefetch size. In addition, subscriptions can include a given selection based on their headers to restrict the consumption to a subset of the messages.

3. Use cases for ATLAS Distributed Data Management

In the following, notifications via both queues and topics have been considered to improve the performance of DDM operations (see Figure 1).

Notifications via topic are able to inform multiple clients about the state of the ATLAS datasets. The use of such topics is experimented or considered by the following subscribers:

- The ATLAS Metadata Interface (AMI [4]), which associates metadata to the datasets, and could populate its catalogue using notifications. This implementation is currently being tested.
- The Production and Distributed Analysis (PANDA [5]) system, which manages the processing of datasets in the ATLAS computing grid. For example, notifications could inform PANDA about dataset availability on the grid.
- Any single ATLAS analyzer, who could trigger his analysis on new datasets by subscribing to a topic, thus enabling automatized analysis shifts.

The ATLAS DDM operations rely on an Oracle database. In order to avoid performance drops due to concurrent access and database polling, notifications via queues have been implemented for an enhanced load balancing, as described in Section 4.

In Section 5, it is shown that the insertion of new elements in the DDM database from external applications can be smoothed by interfacing the transactions with queues.

Figure 1. Representation of the different use cases of message queuing for the ATLAS Distributed Data Management.
4. **Use of notifications via queues to avoid database polling.**
A generic case is considered in this Section. Entries stored in the DDM database may be associated to tasks that have to be executed by external agents. Such elements are typically characterized by the states: waiting, running, or done. A waiting time is defined as the time required for an agent to get the task from the database and begin processing it. Examples of this scheme can be found in services dealing with the tracking of the datasets, where a consistency check can be requested following the deletion of a dataset.

In order to reduce the load on central DDM services, the waiting time needs to be improved. For this purpose, two methods are compared. Tests are carried out using either SQL queries exclusively or SQL queries in association with message queuing to avoid database polling, as described in Figure 2.

**Figure 2.** Representation of the consumption of tasks stored as entries in the DDM database. Requests are read out by agents using SQL queries exclusively or SQL queries in association with message queuing.

For a given agent, the following workflow is applied:

1. Select waiting tasks.
   - case ActiveMQ: the database table containing the tasks sends notifications to a queue via a database trigger, while an agent subscribes to this queue
   - case Oracle: an SQL query picks up the tasks
2. Update tasks to state 'running' via SQL query
3. Process task (in these tests, the process is a pause of 1 ms)
4. Update tasks to state 'done' via SQL query

The implementation of ActiveMQ together with SQL queries for updates constitutes an intermediate hybrid solution to favor robustness. The performances are evaluated with stress tests, as no waiting time is imposed when populating the queue by dispatching events from the database. Moreover, the agents constantly ask for tasks and handle them one by one. In
addition, the table of tasks in the database is not cleared between the tests to simulate real conditions.

The use of message queueing results in a better partition of tasks between the agents, as shown in Figure 3. An improved performance in terms of waiting time is also observed, which is stressed by Figures 4 and 5.

![Figure 3. Repartition of the tasks among the 16 agents that are getting the requests either via message queuing (ActiveMQ) or SQL queries (Oracle). A total of 8000 requests are considered for this test.](image)

![Figure 4. Average waiting time to get tasks as function of the number of parallel agents getting the requests either via message queuing (ActiveMQ) or SQL queries (Oracle). A total of 8000 requests are considered for this test.](image)

![Figure 5. Average waiting time to get tasks as function of the number of requests to be processed. 8 agents are getting the requests either via message queuing (ActiveMQ) or SQL queries (Oracle).](image)

5. Use of notification via queues to improve insertion of new entries in database

This Section focuses on another utilization of notification via queues. Tests are carried out on an external tool that communicates with the DDM database, namely the DQ2 Tracer [6].

Operations on files are traced by this DDM tool so as to report on the type of operation, the site and dataset concerned, or the rate of a transfer. These information are then further processed for analysis, which make it possible, for example, to estimate a dataset popularity. A huge number of operations on files are observed, of about 3.6 million per day, with an average
of nearly 50 per second (see Figures 6 and 7). Therefore, concurrent one-by-one insertions of traces into the database dramatically affect its performance.

![Figure 6](image1.png)  ![Figure 7](image2.png)

**Figure 6.** Number of traces sent as function of the time (in hours), within one day.  **Figure 7.** Number of traces sent as function of the time (in seconds) within 5 minutes (with an interval of 1 second).

The insertion of traces is currently managed via http requests. A new solution is to take advantage of ActiveMQ notifications via a queue. The tests that have been done only exercise the sending of traces via http and notifications to the Apache and ActiveMQ servers, respectively (left part of the diagram in Figure 8).

![Figure 8](image3.png)

**Figure 8.** Representation of the sending of traces from the tracer client to the DDM database. Traces are sent either via http post or via notifications.

In the test protocol, 5 concurrent processes of 200 threads are running during 300 seconds. We recorded 85176 http transactions, with about 300 transactions per second, and 1690 traces lost. When using notifications, 278827 transactions were recorded, with about 900 transactions per second, and no trace lost. Figures 9 and 10 provide the number of transactions as function of the duration of the test, in the case of http posts and ActiveMQ notifications, respectively.
6. Conclusion
The implementation of message queuing technologies using the message broker ActiveMQ showed an improved load balancing in the case of multiple requests to the database. Furthermore, notifications via message queuing proved to be efficient when utilized as an interface to populate the DDM database from external agents. Besides, notifications turn out to be a convenient way to inform external applications about the state of the registered datasets.

References
[1] M. Branco, et al, Managing ATLAS data on a petabyte-scale with DQ2, J. Phys.: Conf. Ser. 119 062017 (2008).
[2] http://activemq.apache.org/
[3] http://stomp.codehaus.org/Protocol
[4] S. Albrand, J. Fulachier, and F. Lambert, The ATLAS Metadata Interface, J. Phys.: Conf. Ser. 219 042030 (2010)
[5] T. Maeno, PanDA: distributed production and distributed analysis system for ATLAS, J. Phys.: Conf. Ser. 119 062036 (2008)
[6] M. Lassnig, T. Fahringer, V. Garonne, A. Molfetas, and M. Branco, Stream monitoring in large-scale distributed concealed environments, in the 5th e-Science conference, Oxford, Dec 9-11, IEEE Computer Society (2009).