High-Performance Scalable Information Service for the ATLAS Experiment

S. Kolos[1], G. Boutsioukis[2], R. Hauser[3]

[1]University of California, Irvine, USA
[2]Aristotle University of Thessaloniki, Greece
[3]Michigan State University, USA

E-mail: Serguei.Kolos@cern.ch

Abstract. The ATLAS[1] experiment is operated by a highly distributed computing system which is constantly producing a lot of status information which is used to monitor the experiment operational conditions as well as to assess the quality of the physics data being taken. For example the ATLAS High Level Trigger(HLT) algorithms are executed on the online computing farm consisting from about 1500 nodes. Each HLT algorithm is producing few thousands histograms, which have to be integrated over the whole farm and carefully analyzed in order to properly tune the event rejection. In order to handle such non-physics data the Information Service (IS) facility has been developed in the scope of the ATLAS Trigger and Data Acquisition (TDAQ)[2] project. The IS provides a high-performance scalable solution for information exchange in distributed environment. In the course of an ATLAS data taking session the IS handles about a hundred gigabytes of information which is being constantly updated with the update interval varying from a second to a few tens of seconds. IS provides access to any information item on request as well as distributing notification to all the information subscribers. In the latter case IS subscribers receive information within a few milliseconds after it was updated. IS can handle arbitrary types of information, including histograms produced by the HLT applications, and provides C++, Java and Python API. The Information Service is a unique source of information for the majority of the online monitoring analysis and GUI applications used to control and monitor the ATLAS experiment. Information Service provides streaming functionality allowing efficient replication of all or part of the managed information. This functionality is used to duplicate the subset of the ATLAS monitoring data to the CERN public network with a latency of a few milliseconds, allowing efficient real-time monitoring of the data taking from outside the protected ATLAS network. Each information item in IS has an associated URL which can be used to access that item online via HTTP protocol. This functionality is being used by many online monitoring applications which can run in a WEB browser, providing real-time monitoring information about the ATLAS experiment over the globe. This paper describes the design and implementation of the IS and presents performance results which have been taken in the ATLAS operational environment.

1. Introduction

ATLAS[1] is one of the two general-purpose Large Hadron Collider (LHC) experiments at CERN, which has been put into production in 2010. The ATLAS detector is operated by the TDAQ system[2] which transfers data from the detector readout to the mass-storage, reducing the data rate from the initial design 40 MHz bunch crossing rate, delivered by the LHC machine, to a few hundreds Hertz of recorded data. In order to perform this task the TDAQ system utilizes more then 2000 modern computing nodes inter-connected by 1Gbit network lines. Each computing node, used by the TDAQ system, executes multiple software applications for various tasks, like data-analysis for event rejection, full event building, event recording, event monitoring, etc. In order to operate in a coherent way all those components require a low-latency and high performance system for exchanging control and monitoring information in real time. Such a system has been developed and is now successfully used for operating the experiment. The system is called Information Service or IS. The IS is used by the TDAQ components for exchanging their information and at any given moment holds a most up-to-date snapshot of that information. The ATLAS experts and members of the shift crew can use various monitoring GUI applications to retrieve and display that information from the IS.

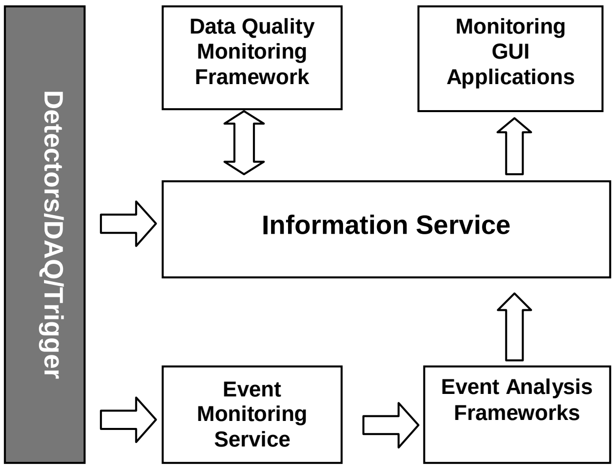

Detectors/DAQ/Trigger
Data Quality Monitoring Framework
Monitoring GUI Applications
Information Service
Event Monitoring Service
Event Analysis Frameworks

Figure 1. The ATLAS online monitoring system architecture.

As shown in Figure 1, IS plays the main role in the information exchange in the ATLAS online environment. The Event Monitoring Service[3] provides statistical samples of physics events which are selected according to the given physics properties, like stream, trigger or sub-detector type. Event Analysis Frameworks produce histograms and publish them to the Information Service(IS). The Data Quality Monitoring Framework[4] retrieves those histograms from the IS, analyzes them with the pre-configured data quality algorithms and publishes the Data Quality results produced by those algorithms back to the IS. All those services as well as all other TDAQ applications publish their operational statistics to IS.

The core of the Monitoring System is the Information Service, which at any given moment holds a snapshot of the most up-to-date monitoring information produced by the experiment. The ATLAS experts and members of the shift crew can use various monitoring GUI applications to retrieve and display specific information from the IS.

2. The Information Service

The Information Service uses client-server architecture, with the Information Repository acting as a server to hold information provided by ATLAS software applications. Currently the Information Repository is implemented by a number of processes, called IS Servers, which are distributed over several computing nodes in a location transparent way. Each server has a unique identifier, which a client application uses to communicate with this server. An IS server holds all information in its memory, saving it on the local hard disk when it is terminated and reloading this information from the disk at the next start.

2.1. The IS Object Model
The IS uses object model for the stored information. Each information object in IS has a type and a value. An information value is represented by a fixed set of attributes where each attribute is of one of the primitive types (like integer, float, double etc.) or of a vector of one of those types. Information type defines the order of the attributes and their individual types. In addition IS supports constructed types for attributes, i.e. an attribute of one IS type can be of another custom IS type. Both type and value are available at run-time via the IS API.

The IS type system supports inheritance. An IS information type 'Derived' may inherit from another IS type 'Base', in which case one can use objects of the type 'Base' for reading information of the type 'Derived'. The type inheritance can also be used for the information selection, as explained in the following sections.

IS types are fully dynamic, which means that each IS client can define its own type of the information and use it at run-time to write and read objects of that type to and from the IS. On top of that IS also provides an access to the meta-information for the available IS types. This information includes the name and human readable description of the type's attributes, which was provided at the moment of this type declaration. In order to support this feature, IS defines the XML format which has to be used for declaring information types and provides the tool which generates the C++ and Java declarations from that XML. The generated code can be used to read and write information objects of the given type, while XML description is used for supporting the meta-type description.

2.2. The Object Identification
Each information object has a unique name in the IS repository. In the current implementation, the name is a character string which must have the following format:

```
InformationName ::= ServerName '.' ObjectName
```

The ServerName must be a valid name of one of an existing IS server application. The ObjectName must be unique for each information object in this IS server.

2.3. The IS Operations
The Information Repository supports three main types of client interactions which are shown in Figure 2. Information Providers can add information to the IS repository as well as update or

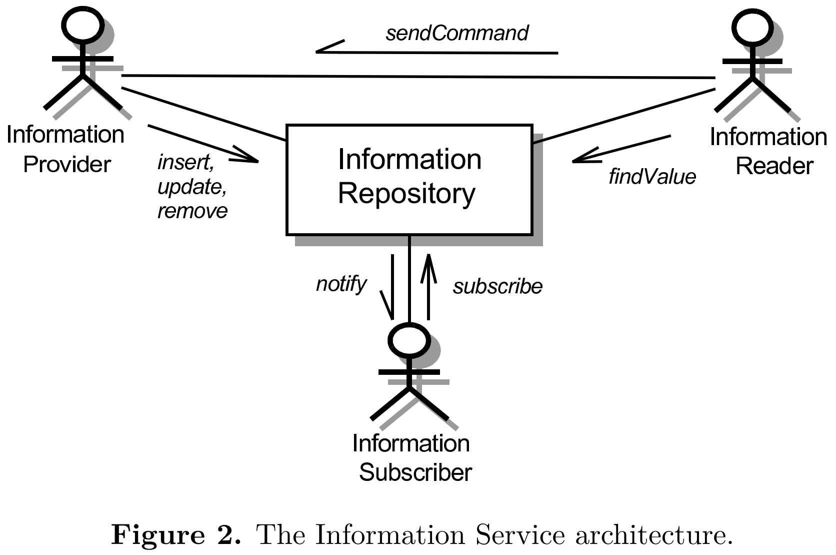


Figure 2. The Information Service architecture.

delete already existing information. Information Readers can retrieve the value of an information from the repository while Information Subscribers can subscribe the repository to be notified about any changes in it. Each time a Provider creates, updates or deletes an information all relevant Subscribers are informed.

In addition to the basic getValue operation, the readers of the information can also use iterators over the selected subset of information. Information can be selected by applying a Posix style regular expression to the information names or by selecting information objects of a certain type. IS API supports several advanced ways for the type based object selection, in particular one can select:

objects of the given type

objects of the given type and of all its sub-types

objects of any type but the given one

objects of any type but the given one and all its sub-types

The type based selection can be combined with the name based one by using 'and' and 'or' logical operations. The same approach is used for the 'subscribe' operation, which allows subscribing for a subset of information, which can be defined in the same way as the iterators subset.

2.4. Inter-process Communication Technology

The current IS implementation is based on the Common Object Request Broker Architecture (CORBA)[5] standard. The C++ implementation is done on top of the omniORB[6] CORBA broker and Java implementation uses the one that is called JacORB[7]. However, both C++ and Java IS APIs are fully independent of the underlying communication layer, thus allowing change to a different communication technology without affecting the IS client applications.

IS also relies on a proprietary CORBA Naming Service[8] implementation, where the CORBA object references to all active IS server applications are held. The Naming Service provides the principal mechanism through which most clients of an ORB-based system locate objects that they intend to use. When an IS server application is started, it binds its name with its own CORBA reference on that Naming Service. Each client, which needs to read or update an information object on that server, has to obtain first its reference from the Naming Service and then send the necessary request to the server using that reference.

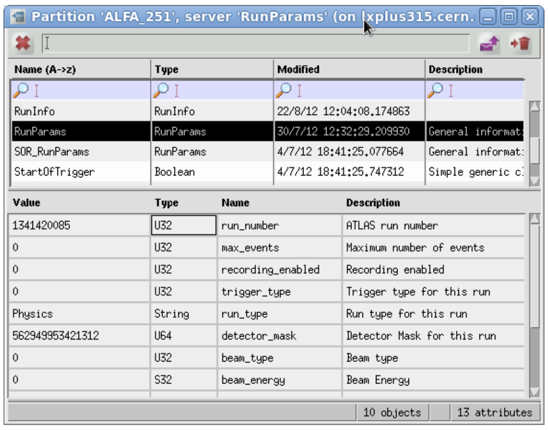
2.5. The IS server

The server functionality of the Information Service are provided by the binary executable, which is called IS server. This is a multi-threaded C++ application which implements the IS CORBA IDL interface and is responsible for keeping all available information in its memory and replying to client requests. The clients' requests are processed by the pool of threads with configurable size. Usually the number of threads in the pool equals to the number of CPU cores on the computer where the IS server process is running. This is the most efficient configuration with respect to the server's performance.

The IS server implements efficient zero-copy scheme for delivering information to subscribers. When an information update request arrives on the server, it delivers it to all subscribers of that information from the same worker thread where the request is being processed. This saves the memory and CPU resources as information copying is not required. This approach works fine as soon as all subscribers are fast enough to process the incoming messages. However, if one or several subscribers are not fast enough, there is a risk that all worker threads will be blocked on delivery of messages to those subscribers. To avoid that, the IS server assigns priority to each subscriber which is a number varying from zero to the number of worker threads in the pool. Initially each subscriber has its priority set to the maximum. This defines the number of concurrent threads which the IS server can use for delivering messages to this subscriber. Each time a slow subscriber condition is detected the priority of this subscriber is decremented, which reduces the number of threads which can be allocated to this subscriber. For slow but alive subscribers the right balance is reached at some priority number and the system keeps working with reasonable message delivery rate. For a hanging subscriber, the priority reaches zero very soon and the IS server starts placing messages to internal queue, which is dedicated to the subscriber. When the queue gets full (it has configurable maximum size) the new messages are dropped. As shown in the last chapter of this paper, this scheme works quite well for for both fast and slow subscribers.

2.6. Programming Languages and Operating Systems
The IS Application Program Interface (API) is available in C++, Java and Python. For the moment the C++ binaries of the Information Service are produced only for the Linux operating system as this is the only OS being used by the ATLAS TDAQ. However, the IS code is compatible with many other platforms, including MacOS, Solaris, HP-UX and few other flavors of the Unix like operating systems.

3. The IS Graphical User Interfaces
Most of the GUI applications, used in the ATLAS online environment, read information from IS and display it in various forms. There are two types of such applications: general displays which can handle any type of IS information by presenting it in some unified way and specialized displays, which can present only a well defined subset of the information from IS in a specific way. In the first group, the most widely used applications are the IS Monitor, which allows to browse the IS repository and shows the value of a currently selected information item, and the Operational Monitoring Display(OMD), which can display the time evolution of the IS information values in a form of time series graphs or a distribution of the values amongst a set of homogeneous IS objects in a form of bar charts. Figures 3 and 4 show snapshots of those applications.



Partition 'ALFA_251', server 'RunParams' (on lxplus315.cern.

Name (A->z)	Type	Modified	Description
RunInfo	RunInfo	22/8/12 12:04:08.174863	
RunParams	RunParams	30/7/12 12:32:29.209930	General informat:
SOR_RunParams	RunParams	4/7/12 18:41:25.077664	General informat:
StartOfTrigger	Boolean	4/7/12 18:41:25.747312	Simple generic c:

Value	Type	Name	Description
1341420085	U32	run_number	ATLAS run number
0	U32	max_events	Maximum number of events
0	U32	recording_enabled	Recording enabled
0	U32	trigger_type	Trigger type for this run
Physics	String	run_type	Run type for this run
562949953421312	U64	detector_mask	Detector Mask for this run
0	U32	beam_type	Beam type
0	S32	beam_energy	Beam Energy

10 objects | 13 attributes

Figure 3. IS Monitor.

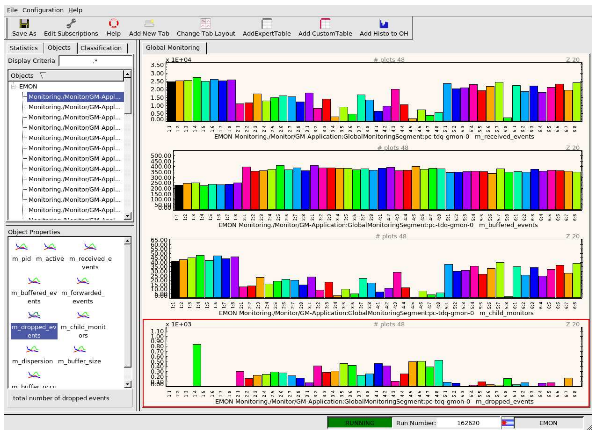

File Configuration Help
Save As
Edit Subscriptions
Help
Add New Tab
Change Tab Layout
AddExpertTable
Add CustomTable
Add Histo to OH
Statistics
Objects
Classification
Display Criteria
Global Monitoring
EMON
Monitoring./Monitor/GM-Appl...
Object Properties
m_pid
m_active
m_received_events
m_buffered_events
m_forwarded_events
m_dropped_events
m_child_monitors
m_dispersion
m_buffer_size
total number of dropped events
EMON Monitoring./Monitor/GM-Application:GlobalMonitoringSegment:pc-tdq-gmon-0 m_received_events
EMON Monitoring./Monitor/GM-Application:GlobalMonitoringSegment:pc-tdq-gmon-0 m_buffered_events
EMON Monitoring./Monitor/GM-Application:GlobalMonitoringSegment:pc-tdq-gmon-0 m_child_monitors
EMON Monitoring./Monitor/GM-Application:GlobalMonitoringSegment:pc-tdq-gmon-0 m_dropped_events
RUNNING
Run Number: 162620
EMON

Figure 4. Operational Monitoring Display.

The second group of the GUI applications contains a number of common tools, used to present some specific monitoring information in real-time fashion, like for example the Data Quality Monitoring Display[9], used to show the results of the online data quality assessment and the Online Histogram Presenter[10], which displays the evolution of online histograms. Figures 5 and 6 show snapshots of those applications.

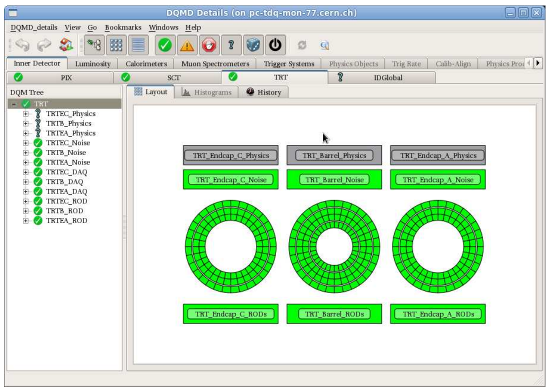


Figure 5. Online DQM Display.

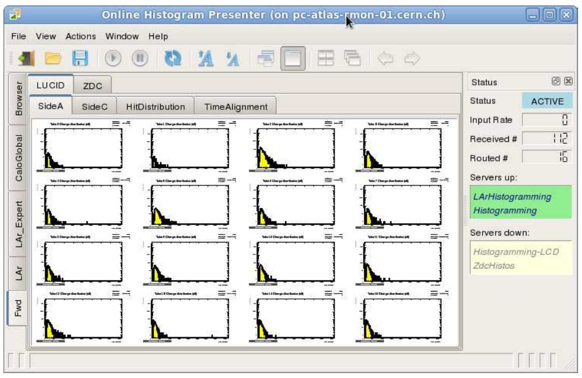


Figure 6. Online Histogram Presenter.

4. The Web IS interface

The IS gives access to every single item in the IS on demand via the HTTP protocol. This feature is provided by the special Python wrapper script, which accepts a special class of HTTP requests (GET URL) and sends back dynamically formed XML text, which contains the value of the IS object, pointed by the given URL. The idea is depicted in Figure 7. The online ATLAS web server provides the interface to the outside world while internally it interfaces to multiple servers written in Python, communicating with them via the Simple Common Gateway Interface (SCGI) protocol. The latter ones are talking to the online IS Repository.

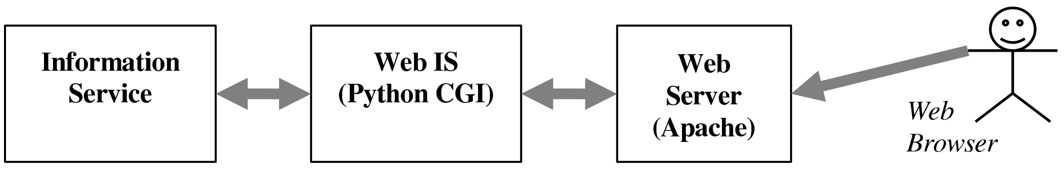


Figure 7. Web interface to the Information Service

In order to get the value of an IS object from a running ATLAS online system, one can put a specially formed URL into the any Web browser. The URL must have the following format:

```
Web-IS-URL ::= 'https://'
  Web-IS-host-name '/'
  Web-IS-prefix '/'
  IS-server-name '/'
  InformationName
```

Here the InformationName is the information ID described in the section 2.2, the Web-IS-host-name defines the name of the Web server where the service is hosted, and the Web-IS-prefix is a special pre-defined token which instructs the Apache HTTP daemon that the given URL must be passed to the Web IS Python server for processing. Below is an example of a Web IS URL, valid for the the actual system setup[1]:

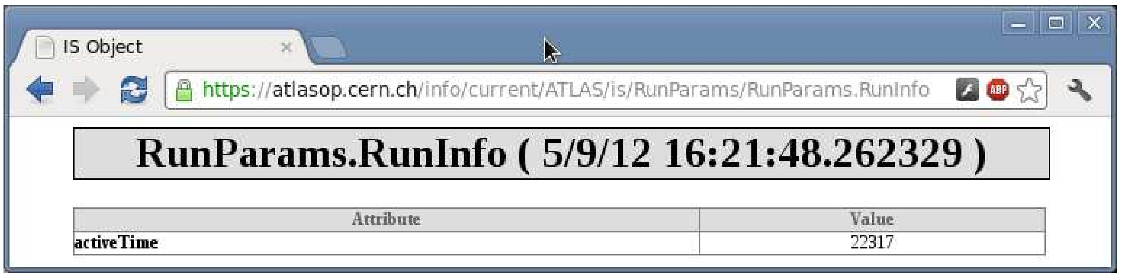
```
https://atlasop.cern.ch/info/current/ATLAS/is/RunParams/RunParams.RunInfo
```

[1] One has to take into account that this facility requires authentication and is available only for the members of the ATLAS collaboration

This URL points to the information object which contains the number of seconds elapsed from the start of the current run. When the Python scripts receives such URL, it extracts the IS information object name from it, reads the corresponding object from IS and sends its representation in XML format back to the Apache server, which uses it to replay to the client, originated the request. As an example, you can find below an XML representation of the RunParams.RunInfo object, received from the Web IS service:

```
<obj name="RunParams.RunInfo" type="RunInfo" time="5/9/12 16:21:48.262329">
 <attr name="activeTime" descr="Integral time the run is active" type="u32">
   <v>22317</v>
 </attr>
</obj>
```

On the client side, one can use CSS, XSL, JavaScript or some other technics to render this data in an appropriate form. Figure 8 shows a result of applying pre-defined XSL template to the RunParams.RunInfo XML data, which was developed to produce table representation of any IS information.


IS Object
https://atlasop.cern.ch/info/current/ATLAS/is/RunParams/RunParams.RunInfo

RunParams.RunInfo (5/9/12 16:21:48.262329)

Attribute	Value
activeTime	22317

Figure 8. IS information object representation in Web Browser.

The system provides authentication, caching and proxying of the information via widely used web industry standards and can be accessed via normal browsers or from any programming language that supports HTTP. For histograms, the server side can do the rasterization of images using the ROOT libraries and return those in a number of standard formats (PNG, JPEG, GIF). Therefore there is no need for any HEP specific software on the client machine.

This design allows users to present the information they are interested in their preferable form. It strikes a balance between having to copy all available information to the outside world, even if it is not used, and having only a predefined static sub-set available. The latency to access an item is determined by the HTTP access, typically in the order of 100 ms if one is outside of CERN. There are limits on the maximum number of concurrent requests, enforced by the servers in order to control the load on the online monitoring system, which otherwise might impact data taking.

Client usage can be as easy as including an *IMG* tag into a web page where the source is a URL pointing to the web service. More complicated scenarios include full browsers for all information written in JavaScript. Scripts are used to regularly access information and post-process it outside of the ATLAS online network. That is very useful for providing sub-system specific monitoring view, which requires a sub-set of monitoring data available for the whole experiment. Different sub-systems may have different ways of post-processing and presenting their specific information.

5. IS Streaming

The IS Streaming facility provides real-time copy of the online monitoring information from the master IS Repository to its mirror counterpart. The information is always passed one-way, from master to mirror, in order to minimize the impact to the master repository. The delay of information transfer is at the order of few milliseconds, so the users of the mirror IS repository practically have the same information snapshot as the ones which are connected to the master repository. The 'mirror' Information Service provides the same API as the master one so all software applications which are capable of interacting with the master IS, will work in the same way with the mirror one. This makes this facility very attractive for supporting the ATLAS Remote Shifts, described in the following section, as it involves no additional learning curve for the shifters and at the same time has zero maintenance cost for the software developers.

5.1. Remote Shift facility

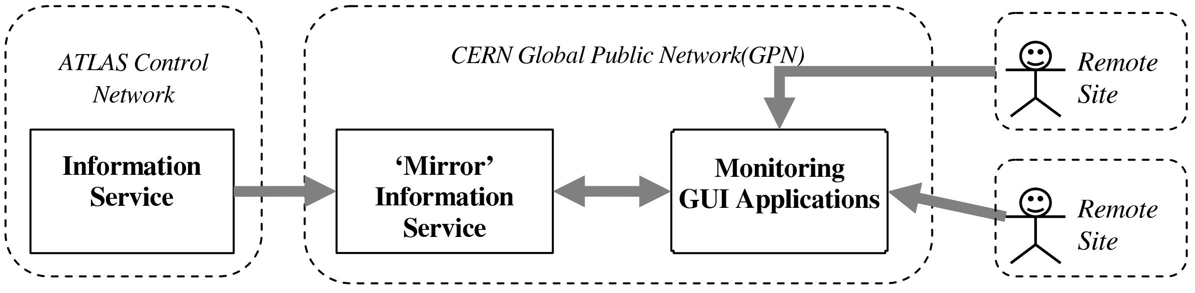

ATLAS Control Network
Information Service
CERN Global Public Network(GPN)
'Mirror' Information Service
Monitoring GUI Applications
Remote Site
Remote Site

Figure 9. Using IS Streaming for Remote Shift facility

In a typical use case, shown in Figure 9 the master repository is the one used by the ATLAS TDAQ system and the mirror is running in the CERN Global Public Network (GPN), where it can be used by a limited number of users for the purpose of performing shifts outside the ATLAS control room. Special network configuration restrictions are applied to prevent any kind of network connections from the 'mirror' monitoring nodes to the nodes inside the ATLAS control network.

6. IS performance measurements

A performance analysis of IS was conducted on the same hardware that was used by the ATLAS production system at the time of writing. The IS was running on a host with two 4-core Intel Xeon E5420 CPUs using the default IS configuration with the number of worker threads set to 8.

6.1. Throughput

The maximum throughput is measured using the maximum delivery rate under optimal conditions, which is the sum of all the messages received by the subscribers in each second. The maximum rate presented on Figure 10, although theoretical, should be representative of the overall efficiency of the implementation; it can be thought as the rate of internal queue operations executed each second by the server application. To reach the maximum rate rate, a configuration of 3 sending processes and 100 subscribers were used. The sending processes were emitting messages at a combined rate of just over 2000 messages per second, while the total delivery rate was recorded to be about 200KHz.

In the next test, IS performance was compared against ActiveMQ[11] and Qpid[12] systems. ActiveMQ and Qpid are general-purpose messaging systems with publish-subscribe functionality very similar to the one provided by IS. However, it must be stressed that the design requirements and purpose of these systems are somewhat broader than those of IS.

Figure 11 shows the delivery rates for equivalent implementations of the same error reporting interface over the above three publish-subscribe systems. The message payload for those tests

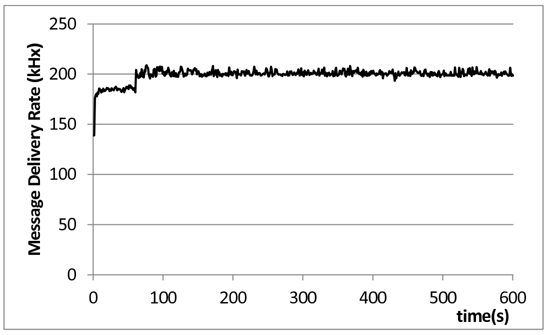


Figure 10. The maximum delivery rate achieved by IS for a message payload size of 9 bytes.

was 50 bytes. For the IS, to reach the maximum rate, a configuration of 12 sending processes and 44 subscribers was used. In order to reach the maximum delivery rate for both Qpid and ActiveMQ it was necessary to use more senders and smaller number of receivers. For the Qpid tests 18 receivers and 40 senders were used, and for the ActiveMQ - 15 receivers and 12 senders. The disparity between the rate of delivery of IS and the two other evaluated systems can be attributed to the much larger message header sizes used by their more complex, general purpose protocols.

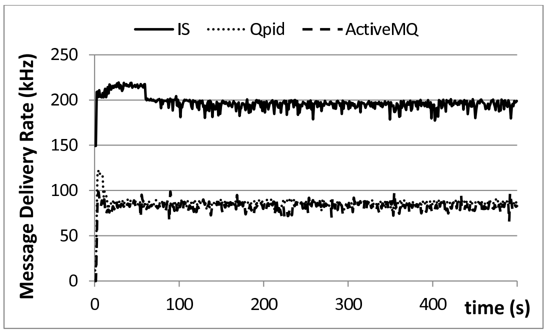


Figure 11. The maximum message delivery rate for IS, Qpid and ActiveMQ with payload size of 50 bytes. The rates of the Qpid and ActiveMQ are practically the same, so the lines are interleaved.

6.2. *Slow subscriber use case*

A particular case that highlights the design differences between IS and other publish-subscribe systems is a configuration with a slow subscriber. In this problematic scenario, one of the subscribers is accepting messages at a much lower rate than the rest (1/5 of the normal rate in this test). This forces the messaging server to retain the undelivered messages until it reaches a certain queue size limit, at which point a certain policy of flow control will be enforced.

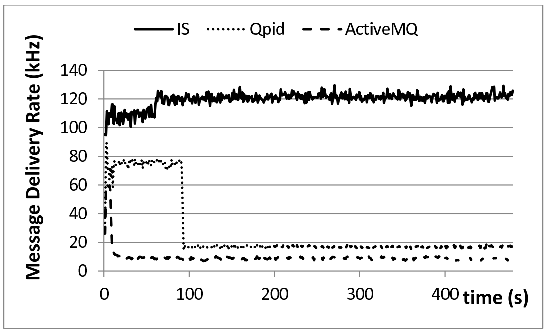


Figure 12. Delivery rates with a slow receiver for IS, Qpid and ActiveMQ. Qpid and ActiveMQ were configured to use a message dropping policy, which IS uses by default.

In this case, the response of a publish-subscribe system can be either to block the producers from sending new messages, or to start dropping some of the messages for the slow receiver.

IS implements the latter policy; when IS reaches a certain threshold of stored messages, it begins to drop the slow subscriber's incoming messages (which does not affect the rest of the subscribers). This is acceptable as Information Service, by design, does not provide a strong guarantee of message delivery. This is one of the reasons why control messages in the ATLAS online environment are not passed via IS.

Both Qpid and ActiveMQ support and use a similar implementation of this policy; however, unlike IS, both systems will sacrifice the system's overall performance in order to minimize the number of dropped messages and to ensure that the messages are dropped in the order they were received. This effect is responsible for the sharp drop of the delivery rate in Figure 12.

6.3. Latency

To evaluate the latency profile of IS, the average latency was measured against two factors that typically have an impact on it, the size of the broadcast message and the number of subscribers to a particular type of message. In both cases the messaging system has to deal with a larger volume of data as the two factors increase, reducing the amount of available resources per message and increasing the time that each message will spend on the messaging infrastructure. Results of the measurements are presented below.

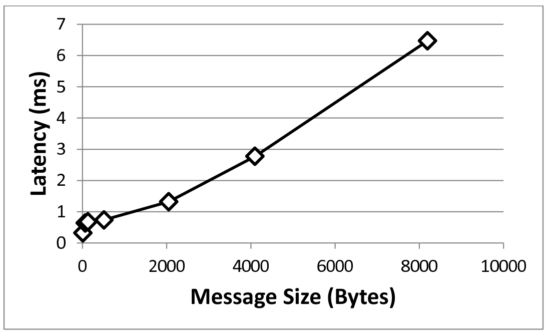


Figure 13. The average recorded latency against the message size.

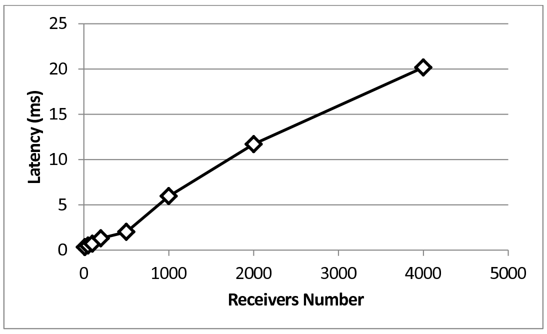


Figure 14. The average recorded latency against the number of subscribers.

The latency on Figure 13 was measured for message sizes between 8 to 8192 bytes, using a fixed configuration of 100 receivers and a single sender emitting messages at a rate of 100Hz, reaching a total delivery rate of 10KHz at all times. For the measurements presented on Figure 14 the size of the emitted messages was set to 8 bytes while a single sender was emitting messages at a frequency of 10Hz. The number of subscribers was in the range of 10 to 4000 subscribers, which translates to the range of total delivery rates from 100Hz to 40KHz.

7. Operational experience

Currently the ATLAS TDAQ system consists of about 10000 processes, each of them publishing some information to the IS. The minimal information, published by any single process is the one that reflects its state. In addition many applications publish some extra information which represents, for example, the status of the hardware those processes are controlling or the the quality of the collected physics data. In the end the total amount of information available in the IS for an average data taking session is at the order of few tens Gigabytes. This information is periodically updated with the update period varying from 5 to 100 seconds for different information types. This information is handled by 260 IS server applications distributed on about 100 computers.

Originally in the first few months of the ATLAS exploitation the some of the IS servers were prone to the slow subscriber issue, when a hanging or half-dead subscriber can slow down one or several IS servers. This issue was affecting the performance of the information update operations, thus slowing down many online applications, communicating with those servers. In order to fix the issue the thread pool strategy, described in the section 2.5 had been implemented. After that fix the Information Service has been working very stably without any major problems.

8. Status and Conclusions

In order to provide efficient information exchange between software applications of the highly distributed Trigger and DAQ system of the ATLAS experiment, the Information Service has been implemented and commissioned in the ATLAS production environment. The service supports arbitrary information types and offers a large variety of access methods for the available information. By using the efficient communication middle-ware, which is based on the CORBA standard, the IS provides very good performance and scalability with respect to the number of information providers and receivers.

Due to its power and flexibility the IS became a central part of the monitoring system for the ATLAS experiment, providing the main source of the real-time monitoring information for the ATLAS shifters in the main control room as well as for the remote shifters who can get the same information via the dedicated remote monitoring system based on the IS information replication mechanism. Experience gained in the two years of the ATLAS production running, establishes the evidence on the IS robustness, while the recent tests, which have been conducted for comparing IS with the other messaging systems available on the market, proves that the IS has a good performance and scaling potential for the future evolution of the TDAQ system.

References

[1] G. Aad et al., The ATLAS experiment at the CERN large hadron collider, Journal of Instrumentation, vol. 3, no. 08, p. S08003, Aug. 2008. [Online]. Available: http://iopscience.iop.org/1748-0221/3/08/S08003
[2] ATLAS high-level trigger, data-acquisition and controls, CERN, Technical Design Report ATLAS-TDR-016 CERN-LHCC-2003-022, 2003. [Online]. Available: http://cdsweb.cern.ch/record/616089/
[3] Scholtes I *The ATLAS Event Monitoring Service - Peer-to-Peer Data Distribution in High Energy Physics* (IEEE Trans. Nucl. Sci. vol 55, iss 1, pp 1610-1620).
[4] Kolos S 2007 *Data Quality Monitoring Framework for the ATLAS Experiment at the LHC* (IEEE Trans. Nucl. Sci. vol 55, iss 1, pp 1-5).
[5] Common Object Request Broker Architecture home, `www.corba.org`
[6] omniORB home, `omniorb.sourceforge.net`
[7] JacORB home, `www.jacorb.org`
[8] OMG Naming Service specification, `http://www.omg.org/spec/NAM/`
[9] Ilchenko Y *Data Quality Monitoring Display for the ATLAS experiment at the LHC*, (2010 J. Phys.: Conf. Ser. 219 022035)
[10] Dotti A *The Online Histogram Presenter for the ATLAS experiment: A modular system for histogram visualization*, (2010 J. Phys.: Conf. Ser. 219 032037)
[11] ActiveMQ home, `activemq.apache.org`
[12] Qpid home, `qpid.apache.org`