A Messaging Infrastructure for WLCG

James Casey, Lionel Cons, Wojciech Lapka, Massimo Paladin, Konstantin Skaburskas
CERN IT, 1211 Geneve 23, Switzerland
James.Casey@cern.ch, Lionel.Cons@cern.ch, Wojciech.Lapka@cern.ch, Massimo.Paladin@cern.ch, Konstantin.Skaburskas@cern.ch

This work has been partially funded by the European Commission under Grant Agreement INFSO-RI-261611.

Abstract. During the EGEE-III project operational tools such as SAM, Nagios, Gridview, the regional Dashboard and GGUS moved to a communication architecture based on ActiveMQ, an open-source enterprise messaging solution. LHC experiments, in particular ATLAS, developed prototypes of systems using the same messaging infrastructure, validating the system for their use-cases. In this paper we describe the WLCG messaging use cases and outline an improved messaging architecture based on the experience gained during the EGEE-III period. We show how this provides a solid basis for many applications, including the grid middleware, to improve their resilience and reliability.

1. Introduction to messaging

Messaging is a method of communication between software components. A messaging client connects to a messaging infrastructure that provides interfaces which allow creating, sending, receiving and reading messages.

Figure 1. The Main Messaging Concepts
Figure 1 summarizes the main messaging concepts:

- a producer (P) sends one message to the messaging infrastructure
- this message reaches in the end n consumers (C1..Cn)
  - if n == 0, the message is in fact dropped
  - if n > 1, the message is duplicated and each consumer receives a copy
- this does not necessarily happen synchronously: the message could stay in the messaging infrastructure cloud for some time before a consumer eventually consumes it.

Distributed communication, enabled by messaging, is loosely coupled. The sending and receiving components are independent and do not need to be available at the same time. The producer doesn’t need to know anything about the consumer and vice-versa. The only things which are required to be known by a producer and a consumer are a message format and a destination.

1.1. Messaging models
There are two main models frequently used: point-to-point and publish/subscribe.

1.1.1. Point-to-point. Point-to-point (PTP) uses the concept of message queues, senders and receivers. Sender is sending a message to a specific queue from where it’s consumed by the receiver. Messages stay on the queue until they are consumed or get expired.

The main features of PTP messaging are following:
- Each message is received by only one receiver. If multiple receivers use the same queue, messages are usually evenly distributed between them.
- Message delivery is guaranteed.
  - There’s no time dependency between a sender and a receiver. The message will wait in the queue for the consumer.
  - Successful processing of a message is acknowledged by the receiver.

1.1.2. Publish/Subscribe. In publish/subscribe model producers send messages to a topic from where they are consumed by all available consumers. It is similar to radio broadcast.

The main features of publish/subscribe messaging are following:
- Each message can have multiple consumers.
- There’s time dependency between producers and consumers. Consumer can consume only messages that were created after its subscription. In addition the client must be active in order to consume new messages.

1.1.3. Other models. The two main models described above are usually extended with additional, often technology specific, options.

For instance, PTP consumers could use selectors to filter the messages they fetch from queues. JMS has the notion of durable subscribers that relaxes the pub/sub time dependency. Then, other models exist. For instance, ActiveMQ [13] has the notion of virtual destinations that can be seen as a combination of one topic and multiple queues.

Finally, in addition to the standard messaging models that take advantage of brokers sitting between the producers and the consumers, some messaging solutions work without brokers. The most promising broker-less technology is probably ZeroMQ [17].
1.2. Messaging systems

The features usually supported by a messaging system are:

- synchronous and asynchronous messaging;
- authentication to control client's access;
- policies to customize resources and their access;
- support of different protocols;
- high availability;
- high scalability;
- interfaces for monitoring.

1.3. Common messaging use cases

Messaging is very versatile and there are many places where it can be used. Here are the most common messaging use cases. These are pure use cases. In practice, what gets implemented is usually a combination of them:

- **Gather**
  - Large number of nodes send data that is collected in a database for further processing

- **Broadcast**
  - Sensors publish information that can be used by external components such as alarm systems

- **Task Queues**
  - Tasks sent to remote components via a set of queues

- **Remote Procedure Call (RPC)**
  - Similar to task queues but with results coming back to the requestor

1.4. Limitations

Messaging is usually not the best solution for:

- large data transfers: messages should be small, most brokers keep them in RAM
- very high message rates: what can be achieved in reality is one or two orders of magnitude lower than the numbers coming from specific lab setups; here are for instance the recommendations of the EMI Messaging Product Team [3]:
  - WAN & STOMP [18]: stay below 1,000 msg/s
  - LAN & binary protocol: stay below 10,000 msg/s
- time critical applications: brokers do add latency compared to direct communications
- high security environments: messaging adds extra code to be audited and extra services to be secured; brokers usually provide only basic security features like plain text authentication and the rest must be added on top (firewall, message encryption...)

2. Messaging and WLCG

Messaging systems are used since several years in industry (e.g., telecommunication, finance ...) as well as in science world (e.g., LHC controls and monitoring and some parts of the grid like service monitoring ...). This technology has been adopted within the EGEE infrastructure as an integration framework for service monitoring and other operational tools including accounting, ticketing and operational dashboards. It also has been used by the Logging and Bookkeeping system [20], [21].

So far, Apache ActiveMQ has been used for this work as it was seen as the most mature messaging open-source implementation available at the time of implementation (early 2007) that provided the
required support in terms of reliability, stability, scalability and access from clients in multiple languages.

Currently work on messaging is done by the Messaging Product Team [4], [5] of the EMI project [3]. The main objectives of this team are:

- collate messaging requirements from the other EMI Product Teams
- analyse the available messaging solutions with regard to these requirements
- recommend which solution(s) to use within EMI
- provide the necessary additional software (tools, libraries...) to run a messaging service on top of existing software
- provide consultancy, guidelines, best practices, examples, ... to help other product teams with messaging integration

3. WLCG use cases
Here are some of the WLCG use cases.

3.1. Service Availability Monitoring (SAM)
Figure 2 shows the architecture of the Service Availability Monitoring (SAM) framework that is used for monitoring of the reliability and availability of the WLCG computing infrastructure. The distributed infrastructure consists of 28 national level Nagios servers deployed across Europe, Asia and America and 4 HEP experiment Nagios servers (Alice, Atlas, CMS and LHCb). Monitoring servers publish alarms and test results to the Messaging Brokers. Alarms are consumed by alarm and ticketing systems. Test results are consumed by the regional and central Data Warehouses and site level Nagios servers which import the regional results as passive checks [6]. The Central Data Warehouse is also receiving data from Open Science Grid, which is using different monitoring system (RSV) [19].

![Figure 2. SAM architecture](image)

In addition to this Nagios servers are using messaging for collecting results of tests executed on grid worker nodes. In total SAM transfers 1 million messages per day.

Before using messaging SAM was based on web services which loaded data directly to the central Data Warehouse. In case of database or web server outages, the tests results were lost. With messaging, the results will wait on the brokers in case any of the underlying components is down (e.g., database layer, message consumer ...). In addition the same test result can be consumed by more consumers (e.g., Central Data Warehouse, site level Nagios servers), which was impossible when using direct web services.
3.2. APEL
Figure 3 shows architecture of APEL [8], which replaced R-GMA transport mechanism with Message Oriented Middleware [9]. Data is being sent to the broker from where it’s consumed to the central database. Aggregated APEL tests are then being sent via the same message brokers to the National Nagios servers. This use case requires security: messages are encrypted and X509 authentication and authorisation are used.

![Figure 3. APEL architecture](image)

3.3. SWAT
Messaging system is used by Site Wide Area Testing System (SWAT) [12], where 2.6 million messages per day are transferred. This use case is similar to the one presented in Figure 3:

- Distributed tests are sent to the broker from where they are consumed by the central database.
- Aggregated data is then being sent to the SAM repository.

Similar problem, similar solution – this is the beauty of using integration patterns [10] and message oriented middleware.

4. Future

4.1. New use cases
Messaging technologies will be used or are being evaluated in following places in the WLCG infrastructure:

- Atlas DDM for DQ2 tracing data.
- BDII update prototype [11].
- LFC catalog synchronization (propagation of changes between LFC catalogs and SE)
- CERN batch service - Analyze usage of resources (CPU, IO, Network, ...) by the experiment jobs on 35000 CPU cores.

4.2. Evolution
The messaging technology ecosystem is changing at a fast pace. In 2010, the AMQP Working Group [14] released a new messaging standard (AMQP 1.0) that looks promising. The same year, VMware bought RabbitMQ [15] that is now a key component of their cloud strategy. In parallel, Red Hat is pushing Qpid which is the messaging technology used in their Red Hat Enterprise MRG product [16]. ActiveMQ also is changing fast with a new version (ActiveMQ 6, codename Apollo) rewritten from scratch in the past months and announced for early 2011. It is not clear yet what will be the best messaging technology for the coming years.
The new use cases depicted above also bring new requirements, especially for reliability and scalability. More messages and more messaging clients require more brokers. From what we learned with WLCG messaging, it is preferable to use separate dedicated services, tuned for different applications, rather than a monolithic "one size fits all" service.

Putting the two together, the future of WLCG messaging will very likely contain different, loosely coupled, maybe heterogeneous, messaging services.

The EMI Messaging Product Team will test the new messaging technologies to recommend which one(s) to use for WLCG. Of course, backward compatibility and interoperability are very important. Several ideas are being tested, including a filesystem based message queue, to allow simple components to be completely isolated from the choice of messaging technology on the broker side.

5. Summary
Messaging is already used or being evaluated in several places in the WLCG infrastructure. It is a robust and effective transport layer which is designed for distributed components and simplifies the architecture. Message Oriented Middleware is a key technology for WLCG because it’s a solid basis for improving the resilience and reliability of grid applications.

References
[1] https://tomtools.cern.ch/confluence/display/MIG
[2] https://tomtools.cern.ch/confluence/display/MIG/Technology
[3] EMI, European Middleware Initiative, http://www.eu-emi.eu/
[4] EMI Product Teams, http://www.eu-emi.eu/product-teams
[5] https://twiki.cern.ch/twiki/bin/view/EMI/EMIMessaging
[6] D. Collados, J. Shade, S. Traylen, E. Imamagic, “Evolution of SAM in an Enhanced Model for Monitoring WLCG Services”, International Conference on Computing in High Energy and Nuclear Physics (CHEP’09), Prague, Czech Republic, March 2009.
[7] Operations Automation Strategy document: https://edms.cern.ch/document/927171
[8] APEL, Accounting Processor for Event Logs, http://goc.grid.sinica.edu.tw/gocwiki/ApelHome
[9] A Distributed CPU Usage Accounting Infrastructure Based on ActiveMQ Messaging System, EGEE Conference 2009, http://goc.grid.sinica.edu.tw/gocwiki/ApelMilestones?action=AttachFile&do=get&target=APEL_EGEE09_POSTER
[10] Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Hohpe G., Wolff B, Addison-Wesley, 2003, http://www.eaipatterns.com/
[11] L. Field, „Designing the Next Generation Grid Information Systems”, CHEP 2010, Taipei
[12] SWAT, https://tomtools.cern.ch/confluence/display/SAM/Site+Wide+Area+Testing+(SWAT)
[13] ActiveMQ, http://activemq.apache.org/
[14] AMPQP, Advanced Message Queuing Protocol, http://www.amqp.org/
[15] RabbitMQ, http://www.springframework.com/products/messaging
[16] Red Hat Enterprise MRG, http://www.redhat.com/mrg
[17] ZeroMQ, http://www.zeromq.org/
[18] STOMP, https://tomtools.cern.ch/confluence/display/MIG/STOMP
[19] RSV, https://twiki.grid.iu.edu/bin/view/MonitoringInformation/RSV
[20] L&B with Messaging, http://egge.cesnet.cz/cvsweb/LB/LB_messaging_poster_UF11.pdf
[21] History of Finished Jobs over Messaging, http://egge.cesnet.cz/cvsweb/LB/job_history.pdf