RUNNING A RELIABLE MESSAGING INFRASTRUCTURE FOR CERN'S CONTROL SYSTEM

F. Ehm, CERN, Geneva, Switzerland

Abstract

The current middleware for CERN's Controls System is based on two implementations: CORBA-based *Controls MiddleWare* (CMW) and *Java Messaging Service* (JMS). The JMS service is realized using the open source messaging product *ActiveMQ* and had became an increasing vital part of beam operations as data need to be transported reliably for various areas such as the beam protection system, post mortem analysis, beam commissioning or the alarm system. The current JMS service is made of 18 brokers running either in clusters or as single nodes. The main service is deployed as a two node cluster providing failover and load balancing capabilities for high availability. Non-critical applications running on virtual machines or desktop machines read data via a third broker to decouple the load from the operational main cluster. This scenario has been introduced last year and the statistics showed an uptime of 99.998% and an average data serving rate of 1.6GByte per minute represented by around 150 messages per second.

Deploying, running, maintaining and protecting such messaging infrastructure is not trivial and includes setting up of careful monitoring and failure pre-recognition. Naturally, lessons have been learnt and their outcome is very important for the current and future operation of such service.

THE CERN CONTROL SYSTEM

Today, the CERN control system is constructed as a three-tier architecture with real-time processes reading signals from the equipment, data processing services and graphical interfaces to display the data.

The signals produced by the accelerator equipment are read out by real-time C++ processes running on around 1600 so-called *Front End Computers* (FEC) and published further to higher level services where it is filtered, evaluated or correlated with other data. The elements in the last and highest level of this architecture are *Graphical User Interfaces* (GUI) mostly written in Java which display the data or let operators in the *CERN Control Centre* (CCC) take direct control on certain equipment by enabling them to set hardware parameters.

All involved components communicate via the Controls MiddleWare (CMW) to publish and/or exchange data. It is actually composed of two products: an in-house developed CORBA [1] based solution (RDA) and ActiveMQ [2], an open source implementation of the *Java Messaging Service* (JMS) [3] API.

RDA is implemented in C++ and Java and provides the point-to-point communication between the involved peers using a physical connection. This allows low latency with a high grade of control. The network and CPU load caused by distributing the data to the recipients resides on the publishing instance.

ActiveMQ is written in Java and was released the first time in 2004 and is in use at CERN for Beam Controls since 2005. It implements the JMS 1.2 specification and fulfils the *Message Oriented Middleware* paradigm where the communication between sender (*producer*) and the recipient (*consumer*) is relaxed by the introduction of an intermediary component (*messaging broker*). By using the JMS API application modules can be distributed over heterogeneous platforms and thus reduces the complexity of developing applications that span multiple operating systems and communication protocols. Producers publish data to the brokers and consumers register their interest in specific messages via the Publish-Subscribe (*Topic*) or Point-to-Point (*Queue*) mechanism. It is then the broker's responsibility to distribute the data reliably.

As a result of this intermediary broker and due to the additional network hop the communication latency increases. However, for the involved systems of the CERN Controls System this latency is acceptable.

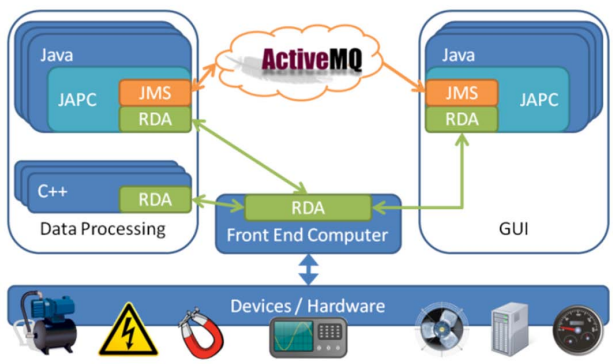

Figure 1: Usage of the Control System Middleware.

As shown in Fig. 1 the ActiveMQ infrastructure is deployed for service to service and service to GUI communication. CMW enables direct communication such as sending commands, tuning hardware equipment and monitoring their parameters. Although not required, data processing services using JMS are exclusively implemented in Java and services which use RDA may be C++ processes as well. In both cases the *Java API for Parameter Control* (JAPC) is used to abstract the underlying middleware.

Choice of a Centralized Messaging System

The main reason for having a messaging system such as ActiveMQ acting as a relay is the clear advantage of outsourcing the work of data distribution to a dedicated entity which is actually made for such a purpose. It originates from the stock market system where the same technology is used to serve thousands of consumers. The control system at CERN does not reach this number but shows very similar requirements in terms of flexibility, scalability and robustness.

CURRENT DEPLOYMENT

The current deployment of ActiveMQ installations consists of 14 brokers used for production and 4 for development. Depending on a project's needs either a single broker or a *broker cluster* for failover and load balancing purposes is set up. A cluster consists of usually two or more interconnected brokers which exchange information about producers and subscribers to forward a message if required. In case one of the brokers crashes clients automatically reconnect to another one in the same cluster.

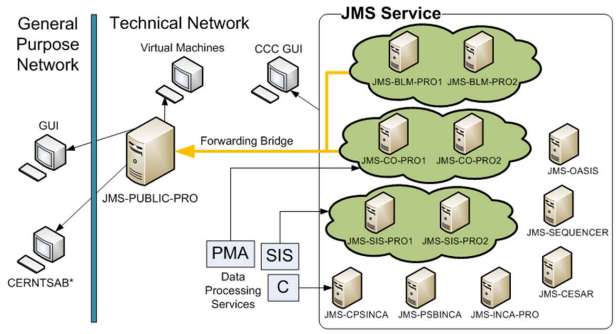

Figure 2: Overview of the Messaging Service.

As Fig. 2 illustrates there are currently 3 main clusters installed for selected middleware services such as the *Software Interlock System* (SIS) and the *Beam Loss Monitors* (BLM) system. The third one is called JMS-CO-PRO and is shared among projects. Alternatively, a project dedicated broker is deployed on the same machine as the related middle tier server. Because these are deployed as single instances it is irrelevant to have a redundant broker service (and hence a higher management load). Assigning

As Fig. 2 also shows that a *data forwarding bridge* to a broker for non-operational clients has been deployed. These clients are GUIs which are not used for beam operations but by developers or experts or as information displays running on windows terminal servers or on virtual machines. Data is forwarded from the main JMS Service to this public read out broker which again redistributes the data to the subscribers. This setup provides several advantages:

- First, a separation of critical clients from non-critical ones allows reducing the additional load caused by latter noticeably.
- Secondly, it enables reading data from outside the closed technical network without exposing the main service machines but a single broker.
- And thirdly, it gives the possibility to instantly remove load from the main service by shutting down the public read broker.

MONITORING

Like other services a messaging broker system needs to be carefully monitored to proactively identify upraising problems and to react to them accurately and immediately. Not only default machine metrics such as network, CPU and disk usage take part in this monitoring activity but additionally specific tools have been developed to collect more – broker specific - information.

Instrument JMX and Sending of a Test Message

One vital prerequisite for such tools is to be able to get an insight view into a running broker. ActiveMQ provides this functionality via the *Java Management Extensions* (JMX) [4] interface and it is easily integrated with existing diagnostic and monitoring tools. Only a subset of the exposed metrics is actually chosen to evaluate the broker health periodically. Information such as memory and storage usage as well as the processed messages since the last iteration is available. However, these numbers may not be sufficient to make a complete statement on the broker condition. Therefore, the *message processing speed* (mps) is measured every 5 minutes to evaluate the time a message takes to go through a broker or a full broker cluster. This is done by an external monitoring agent which sends and reads a test message and compares the submission and receive timestamp. Analogue to the connected applications a high latency is detected very quickly and subsequently treated as a potential problem.

Thresholds are used to determine the severity of the result to then inform service managers via eMail and the *Short Message Service* (SMS). As an example: an mps of 100ms would correspond to a warning, 1000ms to an error.

Topic Monitoring Tool

An in-house developed *Topic Monitoring Tool* (TMT) listens actively to all messages and stores statistics like throughput and size using the *Round Robin Tool* [5] database technology. This again can be used to visualize a history view on the average values like MRTG [6]. An example of the usage of this view is shown in Fig. 3. A producer sending at irregular high data rate was recognized (left red area) and a restart subsequently solved the problem (green area).

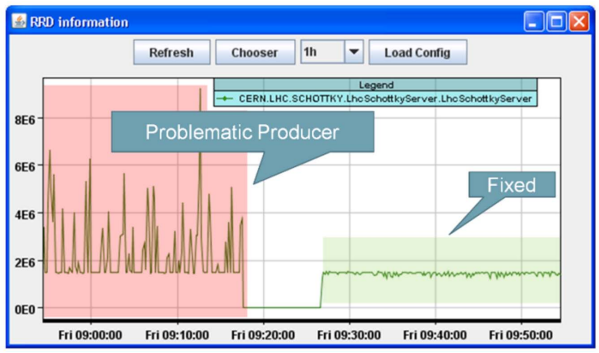

Figure 3: Detecting a producer problem using the TMT.

PERFORMANCE AND RELIABILITY

When it comes to performance and reliability ActiveMQ has proven to be very stable in the present environment and configuration. In 2010 the JMS-CO-PRO and BLM services handled together up to 2.57TByte per day in data volume and reached a service availability of 99.998%. This includes downtimes due to kernel and machine upgrades. Single broker services showed a lower availability in average because of missing redundancy. But 210 consecutive days of uptime with an average of 120 messages per second as one example shows that a single broker instance is very reliable.

Another example for load balancing is the BLM system: it sends out messages of 2 MByte which are read by a large set of consumers. While investigating delay of messages it turned out that the network card in the broker machine was actually a bottleneck. As a consequence, a second broker on a separate machine was deployed to balance the network load. Alternatively, binding the second broker to a second installed network card in the same machine would have had the same effect.

MESSAGE USAGE PATTERNS AND SERVICE LEVEL AGREEMENTS

The usage pattern varies from rather small messages at high frequency (1KByte payload at 100Hz) to large messages at low rate (2MByte payload at 0.5Hz). In case of latter for example, around 25 clients constantly request this data. Other services such as the *Post Morten Analysis* (PMA) system show a very different usage scenario: they send high bursts of small messages only after a beam dump.

Depending on the characteristics of the published data the number of reading clients may vary. In theory, all CCC consoles may read all data at the same time. However, in practice this is not the case. Due to the organization of the CCC an average of 20 GUIs is taken for the data distribution load. This means one message has to be delivered to 20 consumers.

Service Level Agreements

In order to protect a customized messaging service for a project *Service Level Agreements* (SLA) had been established. They describe on a high level the project's needs and message usage patterns and thus help to decide how to realize these requirements. Both sides then have to agree on the SLA and make sure that the rules are respected.

For example: the BLM SLA states that the messaging service must sustain a constant rate of 2MByte payload at 1Hz with 20-30 clients. If much more data is sent than agreed then there is no guarantee that the service will remain stable.

These SLAs also allow tracking of changes of the project's requirements and ease setting up monitoring thresholds. They are based on the experience from the *Large Hadron Collider* (LHC) startup in March 2010 where the amount of data which was estimated was exceeded by a factor of 2.5 (see Fig. 4). A possible messaging service failure was prevented by a precautious upgrade of the machines. This case shows that the prediction for load in such an environment with a great variety of applications (and developers) is not always definitive and final.

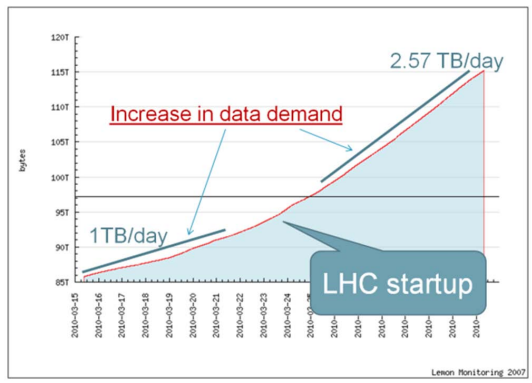

Figure 4: Data demand during start up of the LHC.

PROBLEM ANALYSIS IN PRODUCTION

When having a system which is vital for (beam) operations it is important to be able to quickly analyse a problem to reduce costly downtime. In contrary to a point-to-point communication like CMW a centralized messaging system always faces the problem of being a potential single bottleneck. However, there is also good side. Because of this single point of failure the problem analysis is limited to a fewer places.

For ActiveMQ there are the following instruments for such investigations:

- JMX console showing very detailed information like connections, subscriptions, etc.
- Test clients for reading data from topics
- Changing broker log level without a restart
- Dump of available data via JMX to a SQLite [7] database for easier extraction

The experience shows that most problems caused by an ActiveMQ broker were based on changes which were introduced shortly before (e.g. upgrade or configuration changes).

There is also the much more dangerous possibility that an effect occurs days or weeks later. For example: a new

broker version was tested in an integration test bed before going to production. After the upgrade the service ran smoothly for the next weeks but failed at a certain point due to a bug in the configuration settings of the broker software. As a result, the JMS-CO-PRO cluster was not available anymore to the clients and many critical services stopped working.

Unfortunately, there is no straight solution which is feasible in time and effort for such situations. As for many other services it must be noticed that such incidents may happen (although rarely) despite all monitoring and precautious actions which may reduce the probability but never eliminate such cases.

More important is the ability to know strategies and solutions to quickly resolve the situation.

SUMMARY

ActiveMQ was and is a good choice for decoupled messaging for the CERN's Control System and has proven to be very stable. In particular the strength of scaling linearly makes it inevitable for an environment where the number of reading applications is very dynamic and data demand is growing at the same time. Because unexpected high load is possible it is important to dimension machine resources sufficiently.

In this context, monitoring is a vital part of operation and the evaluation of the recordings must take influence on future deployment decisions. Effective service downtime is reduced by deploying a messaging service per usage domain or project.

It is highly recommended to set up SLAs to track the (growing) user needs. They help to adapt the system to usage scenarios before they are put into production and support setting up monitoring thresholds.

REFERENCES

[1] CORBA, Common Object Request Broker Architecture, OMG, http://www.omg.org
[2] Apache ActiveMQ: http://activemq.apache.org/
[3] JMS Java Messaging Service, http://www.oracle.com
[4] Java Management Extension, Oracle: http://www.oracle.com
[5] Round Robin Tool, Tobias Oetiker: http://oss.oetiker.ch/rrdtool/
[6] MRTG, The Multi Router Traffic Grapher, http://oss.oetiker.ch/mrtg/
[7] SQLite Database: http://www.sqlite.org/