Abstract

The Fermilab Accelerator Control System has recently integrated use of a publish/subscribe infrastructure as a means of communication between Java client applications and data acquisition middleware. This provides a future implementation based on Java Remote Method Invocation (RMI). The DAE implementation of thismiddleware, protected by a firewall, provides services to other languages, and has proven to be much more reliable. A Java client library provides for single synchronous operation as well as periodic data subscriptions. This new system is now used by the synoptic display manager application as well as a number of new custom applications.

Control System

- Single unified control system for the entire complex
  - Including ATLAS and PDG superconducting test accelerators
  - Monitored on ACNET
- Three-tier system
  - Front-end systems attached to field hardware
  - Middleware, database, archiving, and other non-GUI services
  - Applications
    - Access DAEs and their required data through a web interface
    - Application: Java, C++, network, database, etc.
    - Applications run under integrated console environments
  - Custom Java applications not restricted to the control system network
  - Proprietary middleware
  - User-friendly drag and drop builder written in Java

- Data transmitted between middle layer DAEs to Java applications via Java RMI
  - Servlets serve with firewall configuration, load balancing, authentication & authorization, automatic failover in case of DAE failure, etc.

- A new method for this communication based on the AMQP standard has been developed to address all of these issues

AMQP

- Advanced Message Queuing Protocol
- Orginates in financial industry
- Stacked on top of communication protocols
- Enables data transport, queueing, reliable delivery, and security
- Language independent, unlike Java Messaging Service (JMS)
- Different implementations should interoperate, also unlike JMS
- Producers, Consumers, Brokers
- Producers publish messages to an "Exchange" in the broker
- Brokers route messages to "Queues"
- Fanout, direct, round-robin, topic based strategies
- Consumers bind to queues and receive messages
- AMQP 1.0 much simpler and more lightweight
- We use 0.8/0.9.1
- Producers and Consumers only communicate with the broker
- Consumers form a "queue" with the broker
- Exchange is the place where producers send messages
- Messages are routed to queues by the broker based on a routing key
- Hence very simple for threads

Monitoring

- The DAEs make available internal statistics, overall state information, and information about all the requests it is processing via Java Management Extensions (JMX)
- A servlet periodically reads this information and makes it available via a web page
- A user may access data.exe that monitors the performance
- Newer versions of the RabbitMQ broker expose internal information via JMX
- Information is used in the process of configuring and monitoring applications

ACNET Implementation

- Applications, Data Acquisition Engines connect to broker
- Application publishes data request to DAE
- Creates dedicated exchange/queue for reply
- Receives RabbitMQ ticket in message header for authorization
- DAE publishes data on encrypted exchange
- DAE consumes data on the same exchange
- PC: One DAE per broker, currently operates with 3 sets
- Future: One DAE per broker, currently operates with 3 sets
- Overall, we currently don’t use AMQP clustering
-heartbeat mechanism disabled or failed DAE
-Resources request different ones
-Load balancing by random distribution of requests
-Structured data serialized via ACNET Protocol Buffers rather than AMQP

Experience

- This system is now used for all synoptic display manager applications
- A significant number of new custom applications use it
- Other applications using Java RMI have been rewritten
- Overall system performance very well
- RabbitMQ is very reliable and robust
- However, in rare instances when data is sparse and cryptic, the middleware is not able to deliver the data as efficiently as it could
- An additional broker/DAE set was added
- Work is in progress on better load balancing
- Data requests can very quickly in relay the required data throughout
- Although the AMQP 1.0/1.0.1 format we use has been superseded by the very standards compatible AMQP 1.0, RabbitMQ continues to support and enhance their products for this older version.

Conclusions

The Fermilab Accelerator Control System ACNET has recently introduced the RabbitMQ implementation of the AMQP messaging system to transmit data between the middle layer and Java applications. This has involved major changes with the previous method based on Java RMI, such as reliability, firewall configuration, load balancing, and authorization. Work continues on migration to newer versions of the data broker and improved load balancing and monitoring of the system.