The analysis of the performance of RabbitMQ and ActiveMQ

Valeriu Manuel Ionescu
Faculty of Electronics, Communications and Computer Science,
University of Pitesti, Romania
valeriu.ionescu@upit.ro

***Abstract* — Network connected software applications must sometimes transfer data between clients and servers that are not always available. Most popular applications store the data and enable asynchronous communication between clients. One solution to communicate is through Message Oriented Middleware (MOM), more specific through message queues. Message oriented middleware are implemented as services. When MOM-based applications have become popular, the number of message brokers increased significantly. In order for a developer to know what message broker is suitable for his application he needs to compare the available technologies. This paper compares the RabbitMQ and ActiveMQ brokers and presents the results that show the performance difference for both sending and receiving messages.**

***Keywords* — message queues, Message Oriented Middleware, broker, RabbitMQ, ActiveMQ, compare**

I. INTRODUCTION

Message Oriented Middleware (MOM) technologies are very important for building complex software systems. These technologies are used to interconnect independent software systems, for creating a single system. This allows for the networked software systems to be developed using different technologies and platforms that run on different hardware and/or software configurations.

Oriented middleware messages are also used for interconnecting existing with newly developed software or for communication between software systems belonging to different companies.

Message queue is a messaging system that allows applications to exchange data and other information. Critical to the use of message queue in a production system is its ability to scale to meet increasing loads and to continue operating even if failure occurs.

Because of the demand and their widespread use, many middleware solutions have appeared that use message queues to deliver the messages from the source to the destination: some are proprietary software, provided as a service: IBM WebSphere MQ , Microsoft Message Queuing (MSMQ); others are Open Source Software: RabbitMQ, JBoss, ActiveMQ, Qpid, Tarantool; and finally some solutions are based on hardware implementations: Solace Systems, Tervel .

The problem is usually which broker to choose for a specific application. The answer is to research that compares the processing speed for sending and receiving, the memory load, the platforms that brokers can manage, etc.

The current positioning of the Message Queuing servers is in the middle between simple client server communications at one end and complex algorithms used in cloud computing such as Microsoft's Service Bus Messaging platform. The messaging model is presented in Fig. 1.

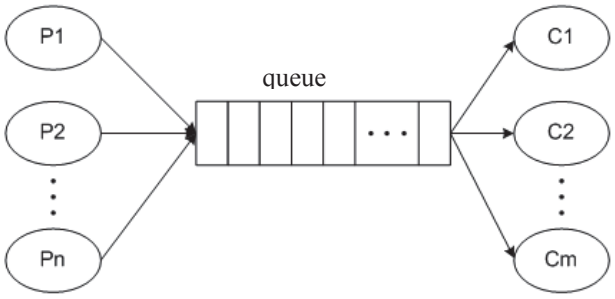


Fig.1. Generic model for sending messages from the producers (P_1-P_n) to consumers (C_1-C_m)

In the messaging queuing model there are multiple specific objects used:

- Connection Factory objects that create the connection to the producer, allowing the connection configuration;
- A connection object is a socket between the clients and the producer. It is used to create one or more sessions (ActiveMQ) or channels (RabbitMQ);
- A producer is an object created by a session (channel) to send data to a destination;
- A consumer is an application that receives messages;
- The queue is a buffer in the server that stores messages. It can be configured to be persistent or not;
- The message is the base data format that is flexible in order to accommodate heterogeneous platforms.

In this paper two of the most used Open Source message brokers are analyzed: RabbitMQ and ActiveMQ.

978-1-4673-8180-2/15/$31.00 ©2015 IEEE

Two Java applications were created that can send and receive text and image messages coded in byte format.

The purpose is to test the impact of the message size on the broker at both sending and receiving message.

Tests were also made to determine the performance dependence on the number of clients.

II. RABBITMQ

RabbitMQ [1-3] is an open source message broker that implements the AMQP standard. It was launched in 2007 and the latest stable version appeared in 2015. It was written in Erlang programming language and based on core framework Open Telecom Platform and from May 2013 controlled the company Pivotal Software.

RabbitMQ has several key features that have contributed to its widespread use.

First, several RabbitMQ servers from a local network can be grouped together, forming one logical broker. This allows the implementation of features such as load balancing and fault toleration.

Another important feature is the AMQP protocol that RabbitMQ uses which accepts connections between different platforms. For example both the message queuing protocol MSMQ used by C# and STOMP that is used by Ruby are accepted.

The implementation of an RabbitMQ application has the steps presented in Fig. 2.

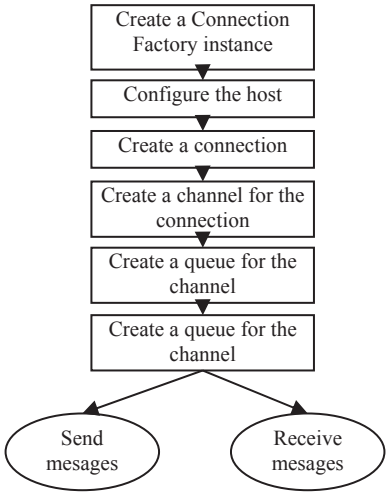


Fig. 2. Basic steps in the creation of a RabbitMQ application

By default, each RabbitMQ sends the messages to the consumers using the Round - Robin algorithm. Also by default, after a message is delivered, it will be deleted from the queue.

III. ACTIVEMQ

ActiveMQ [4-9] is an open source message broker, written in Java, which implements the JMS standard, version 1.1, belonging to the Apache Software Foundation. It is offered with Apache Licensing. The latest version appeared in February 2015.

Many ActiveMQ brokers can collaborate in order to improve the performance scalability. This collaboration type is called brokers network and can support multiple topologies.

Different from RabbitMQ, if a C # application using MSMQ needs to transfer data to an application in Ruby using STOMP, one broker ActiveMQ is not enough to connect these applications. It requires a broker middleware because JMS-based applications can only connect a Java platform to another, not two different platforms. This is the main difference between JMS and AMQP and between ActiveMQ and RabbitMQ.

The implementation of an ActiveMQ application has the steps presented in Fig. 3.

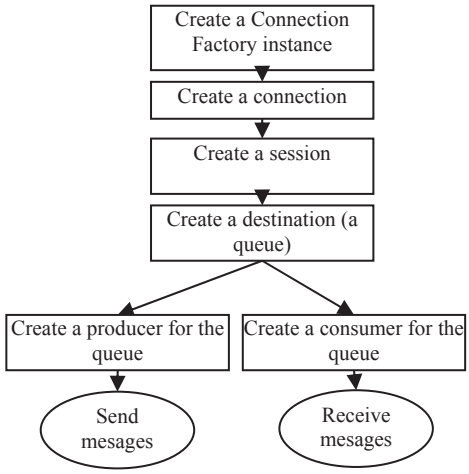


Fig. 3. Basic steps in the creation of a ActiveMQ application

As it can be observed, while the general concepts are similar for both brokers, the implementations differ in fine details.

IV. TEST APPLICATION STRUCTURE

In order to obtain correct results and to increase the implementation and testing speed, two VMWare virtual machines were created on the VMWare ESXi 5.5.0 machine with the same configuration in order to have similar environments. Each virtual machine runs one at a time, in order not to affect the performance of the other machine.

The virtual OS where the brokers run is a Windows 7 x64 on a HP ProLiant BL680c G5 server with a Intel Xeon 7420 at 2.13GHz with 4 processors / 4 threads and 16 GB RAM.

For each broker two applications were created: a client that sends messages and a server that receives messages. The queue stores the messages from the client until the server request them (Fig. 3).

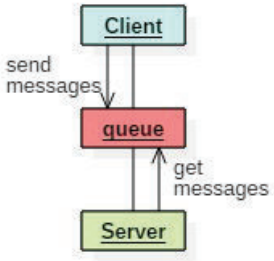


Fig. 4. Collaboration diagram

The client can perform the several actions as seen in Fig. 5.

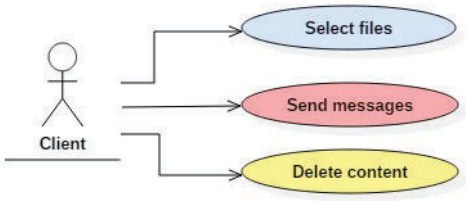


Fig. 5. Use case diagram for Client application

The client only needs to send files. In order to compare the broker's behavior for different file types, the client application has the option of selecting an image or text file. After, it can send the selected messages or, if the file selected is not the right one, it can delete the content of the message.

The server can perform several actions as seen in Fig. 6.

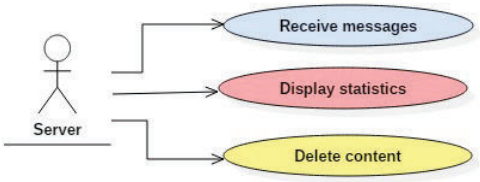


Fig. 6. Use case diagram for Server application

The server will to run continuously. As long as he is running, he can receive messages from the queues, display the queues statistics and delete de content of the messages if the user requests it.

When multiple queues are used, the two applications present particularities for the implementation.

For RabbitMQ, the client application was designed as seen in Fig. 7.

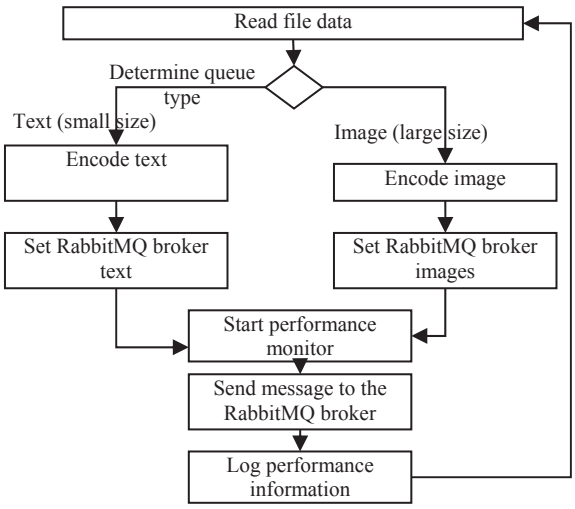


Fig. 7. The activity diagram for the client application - RabbitMQ

When the program starts, it reads the file and determines if it is a binary image or a text message. Afterwords, the message is coded in byte format, and the RabbitMQ broker is set up, then the message can be sent.

In Fig. 8 is presented the activity diagram for the server application.

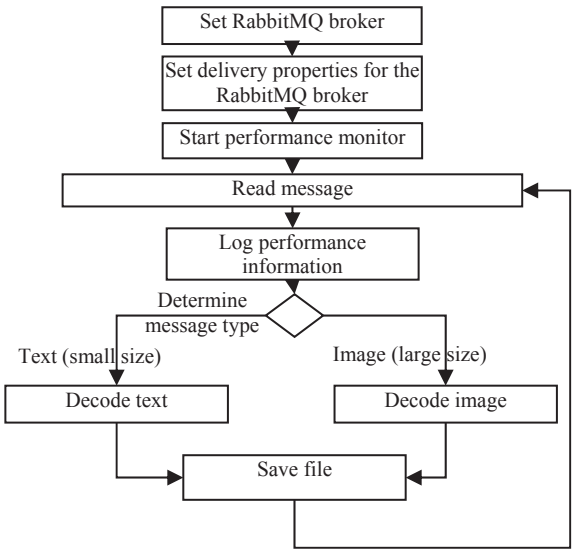


Fig. 8. The activity diagram for the server application - RabbitMQ

After the server starts, it need to be set up, then the program checks if there are messages left in the queues. If the answer is positive, the server receives the messages. For each message, it verifies if it text or image and decode it properly, then save the content.

From the design it can be seen that if there are always messages in the queue, the performance monitor works correctly; if there are no messages in the queue, the first performance monitor will be off, but it is easy to filter as the data is very large compared to the other values.

The time difference was computed with System.nanoTime() function because of the precision needed and the fact that it was necessary to measure a time span and did not need a relation to the calendar time.

For Active MQ the client application was designed as seen in Fig. 9.

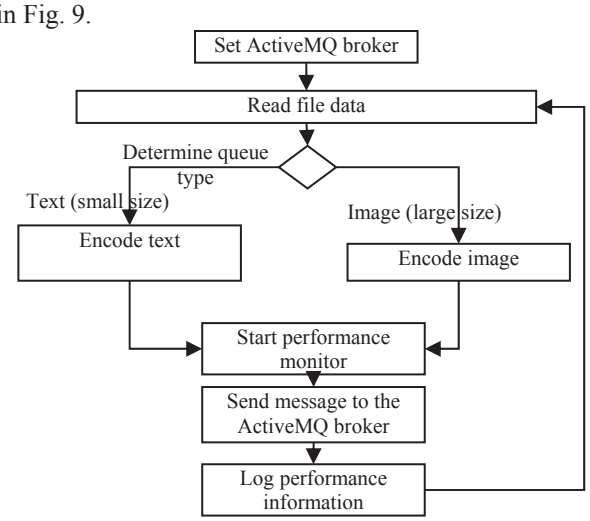


Fig. 9. The activity diagram for the client application - ActiveMQ

This time the ActiveMQ settings will be made before the main program loop. If these settings are made in a loop as was the case for the RabbitMQ broker, errors will be present at queue re-creation and the previous messages will be lost.

The server needs to be configured for ActiveMQ and the main application loop is similar to the previous server.

Message Queuing servers use message persistence so that the queue will survive a server restart. Because this was a test implementation is was not critical to save the data in case the server dies, therefore the function:

producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

was used in order not to persist the messages. This also helped with the tests because in case of errors it was not necessary to check that the queue was empty.

V. Test Resyults

The test system had multiple clients, the broker and multiple applications that retrieve the data from it as presented in Fig. 1. The file sizes for the images used in the tests are presented in the Table 1.

TABLE I. Image Sizes Used in Tests

Size	58KB	148KB	450KB	1.44MB	3.97MB
Width x Height	512x512	1024X1024	2048X2048	4096X4096	8192X8192

In order to shorten the notations, 512x will be used for 512x512, 1024x for 1024X1024, 2048x for 2048X2048, 4096x for 4096X4096 and 8192 for 8192X8192 pixels.

The first test concerned the broker performance degradation when the message size increases.

The RabbitMQ performance for the message sending to the broker:

TABLE II. RabbitMQ Performance for Sending Messages to the Broker

Image	512x	1024x	2048x	4096x	8192x
Message queue placing time (ms)	0.694991	1.706642	3.663944	8.802655	27.794664

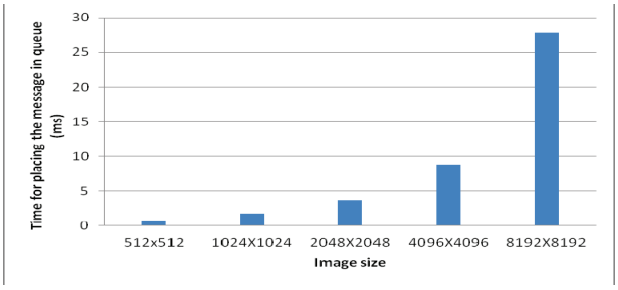


Fig. 10 RabbitMQ performance for sending messages to the broker (lower is better)

The results for ActiveMQ performance for the process of sending the message to the broker:

TABLE III. ActiveMQ Performance for Sending Messages to the Broker

Image	512x	1024x	2048x	4096x	8192x
Message queue placing time (ms)	0.246679	0.715804	1.323073	5.370515	18.150352

The graphical representation of the results is presented in Fig. 11.

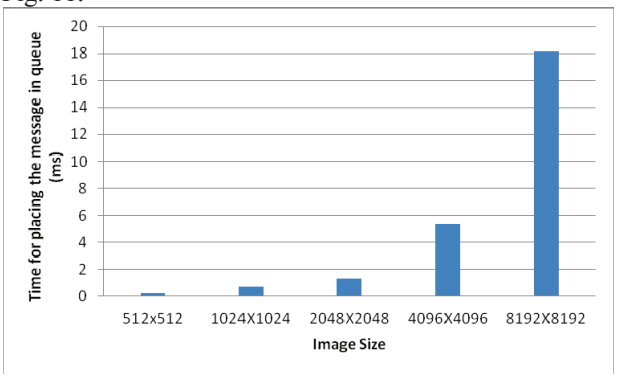


Fig. 11 ActiveMQ performance for sending messages to the broker (lower is better)

The RabbitMQ performance for receiving messages from the broker is presented in Table IV and graphical in Fig. 12.

TABLE IV. RabbitMQ Performance for Receiving Messages from the Broker

Image	512x	1024x	2048x	4096x	8192x
Message queue recieving time (ms)	0.003713	0.004573	0.00662	0.008245	0.012111

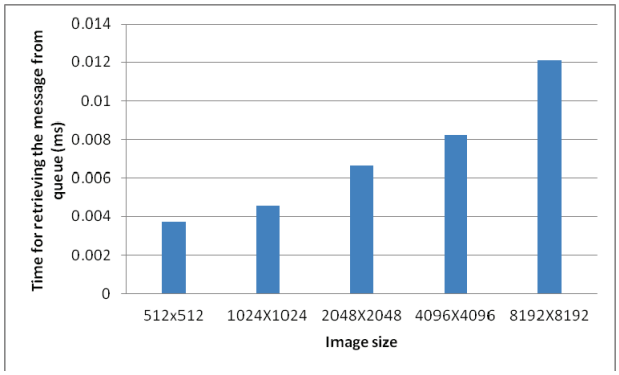


Fig. 12. RabbitMQ performance for receiving messages from the broker (lower is better)

The ActiveMQ performance for receiving messages from the broker is seen in Fig. 13:

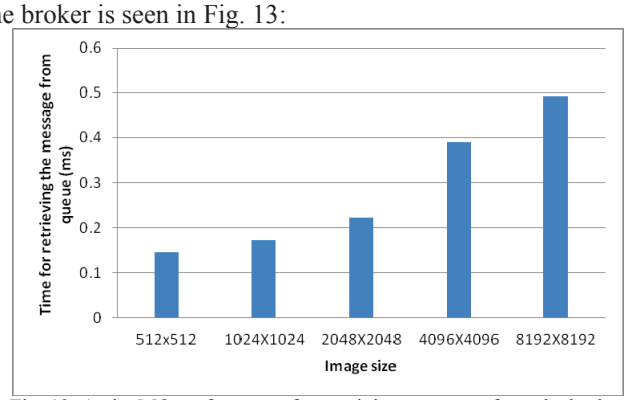


Fig. 13. ActiveMQ performance for receiving messages from the broker (lower is better)

For both of the RabbitMQ and ActiveMQ measurements concerning the performance for receiving messages from the broker, as the servers are really quick, the nanoseconds used were hard to determine and multiple measurements were necessary as various operations of the operation system provoked large (relative to the measured values) variations in the server behavior.

Results comparison is presented in Fig. 14 and Fig. 15.

After testing the broker performance, it is visible that ActiveMQ is significantly faster on message transmission to the broker. But the same sample shows that RabbitMQ is faster on receiving messages from the broker then ActiveMQ, but not significantly.

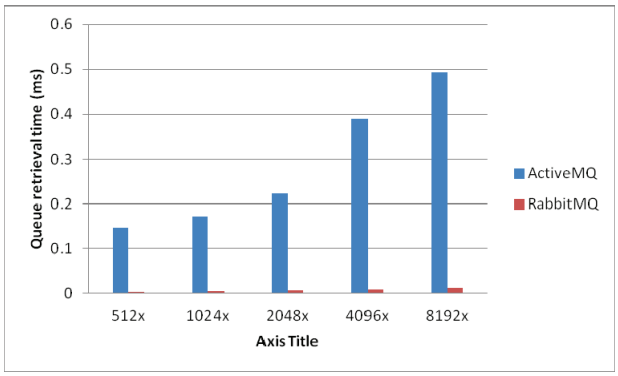


Fig. 14. ActiveMQ vs RabbitMQ performance for receiving messages from the broker (lower is better)

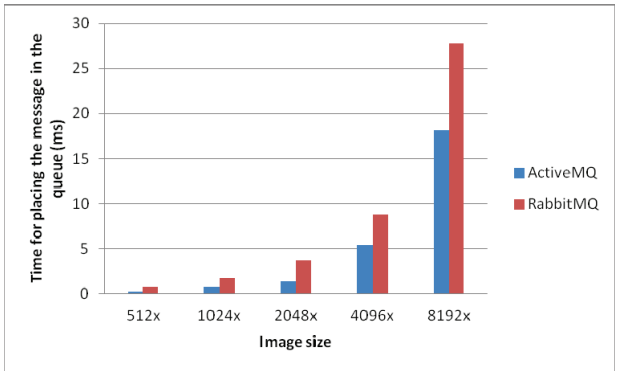


Fig. 15 ActiveMQ vs RabbitMQ performance for sending messages to the broker (lower is better)

A different test involved increasing the number of clients that send data to the broker. As more clients upload data to the broker queue, the performance decreases. Both ActiveMQ and RabbitMQ exhibit a similar behavior.

TABLE V. TIME TO SEND A 1024X1024 IMAGE TO THE BROKER

Number of clients	1	3	10	20
Time per image for ActiveMQ (ms)	1.005885	2.696629	11.23365	33.36614
Time per image for RabbitMQ (ms)	2.245003	2.63496	5.033704	15.69899

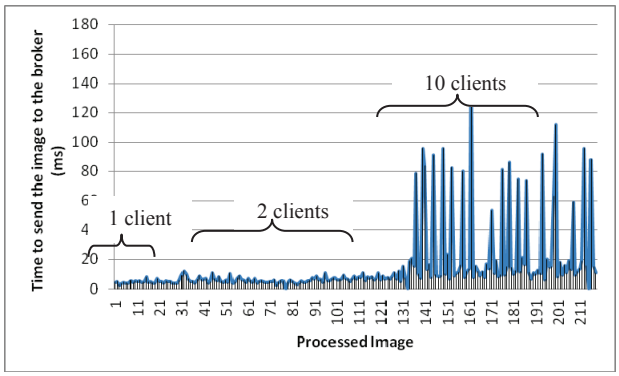


Fig. 16 ActiveMQ broker performance deteriorates as the number of clients increases

The comparison between the same two brokers is seen in Fig. 17 and shows an advantage for the RabbitMQ broker.

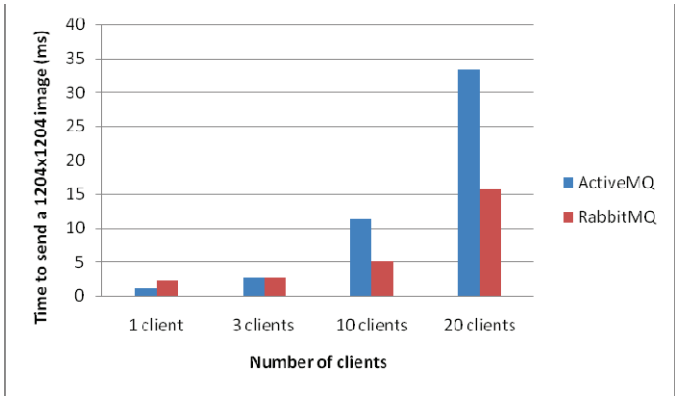


Fig. 17 ActiveMQ broker performance for receiving messagesdeteriorates faster compared to RabbitMQ as the number of clients increases (lower is better)

Other problems that were faced during the implementation and the testing were:

- random lags in ActiveMQ performance if a sustained number of images were sent from the client to the broker. The spikes were not related to other system activities and occurred in a random pattern at several minutes one from another. In comparison, the RabbitMQ server did not experience this behavior.

-when testing the application in the development environment (Eclipse) for the large image size, when the clients were loading data to the broker and there was no server to get the data, a lack of memory error. The solution was to increase the heap size with -Xmx2048min parameter in the application's VM arguments section.

-nanoseconds were too small to accurately measure the performance and multiple runs and manual result filtering was necessary to eliminate erroneous data.

VI. CONCLUSIONS

In this paper two different brokers were compared, RabbitMQ and ActiveMQ, and the performance of each broker is highlighted.

After testing the broker performance, it can be seen that ActiveMQ is faster on message reception (the client sends the message to the broker).

But the same tests shows that RabbitMQ is faster on producing messages (the client receiving messages from the broker) compared to ActiveMQ.

This study is useful when you need to find a suitable broker for an application. For example, a GPS applications for a taxi company that displays the location of its automobiles, MOM services are design to transmit coordinates to a database motor and from there back to administrator when requires. This application would need RabbitMQ, because it sends many messages, but the server is less interrogated.

Another example is credit card payments. Once entered a password, a message leaves in order to check if the password is correct and then an answer is received. Also message queues can be used to confirm payments. This application also could use RabbitMQ, because is faster than ActiveMQ.

In a medical system that records and references networks, MOM technologies are design to carry data to a database stored on a server. When you go to a consult, a doctor, will want to see you medical records before he prescribe some medicine. This application would use ActiveMQ, because of the better performance for loading the messages to the broker.

Other considerations would be the advantage of RabbitMQ that uses AMQP to accept connections between different platforms.

REFERENCES

[1] Alexis Richardson, RabbitMQ – An open source message broker that just work, QCon 2009, Accessed: 20.06.2015, Available at: https://www.rabbitmq.com/resources/RabbitMQcon.pdf

[2] David Dossot, RabbitMQ Essentials, Packt Publishing, April 2014, http://stage.reflectsoftware.com/RMQ/RabbitMQ%20Essentials.pdf

[3] Alvaro Videla, Jason J.W. Williams, RabbitMQ in Action, Accessed: 20.06.2015, Available at: http://nitintech84.weebly.com/uploads/2/1/3/6/21361954/rabbitmqinaction.pdf

[4] Bruce Snyder, Rob Davis, Dejan Bosanac, ActiveMQ in Actions, Manning Publications, Accessed: 20.06.2015, Available at: http://www.zengcode.com/files/FckAjaxFileManage/uploaded/documentations/ActiveMQInAction.pdf

[5] James Strachan, Enterprise Messageing with Apache ActiveMQ, http://people.apache.org/~jstrachan/talks/ActiveMQ-Dublin07.pdf

[6] Mark Richards. Understanding the differences between AMQP & JMS, Accessed 20.07.2015, Available at: http://www.wmrichards.com/amqp.pdf

[7] Oracle, ActiveMQ – Oracle Enterprise gateway Integration Guide, Accessed: 20.06.2015, Available at: http://www.oracle.com/technetwork/middleware/id-mgmt/documentation/activemq-oeg-integration-guide-428892.pdf

[8] Bruce Snyder, Enterprise Messaging With ActiveMQ and Spring JMS, Amsterdam, Apr, 2009, Accessed: 20.06.2015, Available at: http://europe.springone.com/dl/springone-amsterdam-2009/slides/BruceSnyder_EnterpriseMessagingWithActiveMQAndSpringJMS.pdf

[9] Bruce Snyder, Building a Reliable Messaging Infrastructure with Apache ActiveMQ, Accessed: 20.06.2015, Available at: http://www.softwaresummit.com/2007/speakers/presentations/SnyderMessagingWithAMQ.pdf