Computer and Information Science; Vol. 7, No. 4; 2014
ISSN 1913-8989 E-ISSN 1913-8997
Published by Canadian Center of Science and Education

Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware

Sanjay P. Ahuja[1] & Naveen Mupparaju[1]

[1] School of Computing, University of North Florida, Jacksonville, USA

Correspondence: Sanjay P. Ahuja, School of Computing, University of North Florida, Jacksonville, FL 32224, USA. E-mail: sahuja@ospreys.unf.edu

Received: April 25, 2014 Accepted: June 16, 2014 Online Published: August 19, 2014

doi:10.5539/cis.v7n4p9 URL: http://dx.doi.org/10.5539/cis.v7n4p9

Abstract

Message Oriented Middleware (MOM) is an enabling technology for modern event-driven applications that are typically based on publish/subscribe communication (Eugster, 2003). Enterprises typically contain hundreds of applications operating in environments with diverse databases and operating systems. Integration of these applications is required to coordinate the business process. Unfortunately, this is no easy task. Enterprise Integration, according to the authors in (Brosey et al, 2001), "aims to connect and combines people, processes, systems, and technologies to ensure that the right people and the right processes have the right information and the right resources at the right time". Communication between different applications can be achieved by using synchronous and asynchronous communication tools. In synchronous communication, both parties involved must be online (for example, a telephone call), whereas in asynchronous communication, only one member needs to be online (email). Middleware is software that helps two applications communicate with one another. Remote Procedure Calls (RPC) and Object Request Brokers (ORB) are two types of synchronous middleware—when they send a request they must wait for an immediate reply. This can decrease an application's performance when there is no need for synchronous communication. Even though asynchronous distributed messaging using message oriented middleware is widely used in industry, there is not enough work done in evaluating the performance of various open source Message oriented middleware. The objective of this work was to benchmark and evaluate three different open source MOM's performance in publish/subscribe and point-to-point domains, and provide a functional comparison and qualitative study from developers perspective.

Keywords: performance evaluation, Message Oriented Middleware (MOM)

1. Introduction

Message Oriented Middleware (MOM) plays a key role in distributed application development. The integration of applications from a diverse assortment of operating systems and databases is critical for a successful e-business. MOM is used to help applications across multiple platforms communicate with one another, creating a much more seamless business operation. There are different types of commercial and open source MOM's available in the market. Every MOM has its own unique advantages and disadvantages depending on the architecture and the services offered. This paper researches, compares, and evaluates the performance of different open source MOM's from a vendor agnostic perspective. To state simply, MOM delivers messages from a sender to a receiver. It uses queues in the process of delivering messages; for example, a sender application that needs to send a message will place the message in a queue. MOM then takes the message and sends it to the respective destination queue. MOM's are categorized into two types—Point to Point, and Publish/Subscribe.

1.1 Point to Point

This message queuing form is also known as a queuing model. This form will have two main participants: a sender and a receiver. The sender will place the message in the queue, and the respective receiver will receive the message. Once the receiver acknowledges the receipt of the message, the message will be deleted from the queue. There is only one consumer per message, and the sender and receiver have no timing dependencies.

1.2 Publish/Subscribe

This message queuing form has a sender and one or more receivers for a single message. In this form, the sender is called a publisher, because the sender will send the message by publishing a message to the topic. The

receiver(s) subscribed to the topic will then receive the message, which will be present in the topic until all subscribers receive the message or until the message expires. For each type of message in the Publish/Subscribe form of MOM, a "publisher" is chosen which sends out messages, and one or more "subscribers" are chosen which "subscribe" to the messages. Once a subscriber has been registered with the middleware component, any new messages sent by the publisher are automatically delivered to that subscriber in addition to sending the same messages to any listeners, which are already registered, usually through an event or callback mechanism.

1.3 Java Messaging Service

Java Messaging Service (JMS) is an Application Programming Interface (API) provided by Sun Microsystems. JMS API is the part Java 2 Enterprise Edition (J2EE). JMS API is used to develop applications for the underlying middleware provider, providing support for both messaging domains Point to Point and Publish/Subscribe. The typical components present in a JMS application are JMSClient, JMSProvider, and JMSApplication. The JMSClient is an application component that sends or receives messages; the JMSProvider is a middleware component that provides queuing functionality; and the JMSApplication is an application, which consists of clients and one JMSProvider.

Sending a message consists of the following steps:

1) Create: sender creates the message and populates with data
2) Send: transmits the message to messaging system
3) Deliver: the messaging system hands the message over to the receiver
4) Receive: the receiver receives the message from messaging system
5) Process: the receiver process the data contained in the message

The message consists of three parts:

- Header: Information used by both the client and sender to send and receive messages
- Properties: Additional properties of the header those are specific to the application, standard, or to the provider
- Body: Components of the message, which consists of text, object and bytes.

Different types of the message body include:

- Text Message: consists of string or collection of strings
- Stream Message: consists of a stream of Java data types
- Bytes Message: consists of byte arrays
- Object Message: consists of Java objects
- Map Message: consists of a Java map data type

The different open source MOM's that are studied include Open Message Queue (Sun, 2013), Apache Active MQ (Apache, 2013), and Mantaray MQ (Mantaray, 2013).

2. Message Oriented Middleware

This section introduces the three types of Open source Message Oriented Middleware tools used in this study: Open Message Queue, Active MQ and Mantaray MQ.

2.1 Open Message Queue

Open Message Queue (Open MQ) is a community version of Sun Java System Message Queue 4.1, and is implemented using JMS API. It is installed in a central location and allows programs to send messages using the client API.

It further provides enterprise features to support scalability and high availability. The central part of the architecture is the broker, which is the important implementation that is responsible for receiving and delivering the messages. The broker can be clustered for service and data redundancy. The internal communication method between brokers or cluster nodes is carried out by using proprietary methods. Message storage can be achieved by either file store or JDBC data-source. For the purpose of high availability, JDBC data-source is recommended. Open MQ can work directly with JMS over HTTP and can be administered using the built-in Graphical User Interface (GUI) console and Command Line Interface (CLI). Open MQ also supports JMX API for advanced administration. The clients can be programmed in Java using JMS API or C.

2.2 Active MQ

Active MQ is an MQ implementation from the Apache Software Foundation. It supports cross language clients and protocols from Java, C, C++, C#, Ruby, Perl, Python, and PHP and provides support for JMS 1.1 and J2EE 1.4 specifications. This application is designed for high performance clustering, client-server, and peer-based communication.

In the architecture of Active MQ the primary component of the application is the broker, which is responsible for creating and managing network connections. The connector classes handle the connections using different protocols—HTTP, SSL, TCP are all supported. The network service is responsible for being high availability, finding other brokers on the network, and storing and forwarding. Message store is responsible for persistent options using JDBC, file, journaling, and caching of messages. Active MQ provides enterprise features like clustering and multiple message stores and can be installed on any operating system that supports Java.

2.3 Mantaray MQ

Mantaray MQ is a fully distributed peer-to-peer communication and messaging solution. It is developed in Java, and provides Remote Method Invocation (RMI) and JMS API. It also integrates with JBoss (Redhat, 2013), Weblogic (Oracle, 2013) and Websphere (IBM, 2013). Compared to the other two providers, Mantaray MQ employs a fully distributed peer-to-peer architecture. An installation of Mantaray MQ resides on each host on the network, eliminating the bottlenecks of single point of failures. It supports both Point to Point and Publish/Subscribe messaging. The entire configuration is done in the default_config.xml present in the configuration folder of the installation. Information about the other peer is configured uniquely. Many instances of Mantaray can run on the same computer using different ports, and supports different transport types such as TCP, HTTP, and SSL. These transport configurations are configured in the transport section of the configuration file. All peers that are involved in the communications should be configured in world map either statically or using Mantaray's Auto Discovery.

3. Related Work

Tran et al in (Tran, 2002) have evaluated the performance of *IBM's MQSeries V5.2*. This paper gives an overview of the technology and discusses the performance metric used.

Another research reviewed is by the Crimson consulting group (Crimson, 2003). This report is a benchmark comparison of Sun Java System Message Queue 3.5 and IBM Websphere MQ 5.3. The report gives an overview of both the technologies. This paper discusses different testing variables and also discusses about the results.

Chen et al in (Chen, 2004) discuss the relation between JMS and MOM. The authors have presented different metrics for measuring the performance of message persistence. The authors also compared two commercial applications.

Bernstein in (Bernstein, 1996) provides a discussion on different kinds of middleware, evolution, and services offered. He also provided a discussion on different kinds of frameworks and middleware.

As seen from this survey there are papers which have discussed and quantified the performance of commercial MOM's. The literature survey did not reveal any such studies pertaining specifically to open source MOM's. However, the existing papers are valuable in providing a general approach (including suggesting metrics) for this research.

4. Metrics and Research Methodology

This chapter presents metrics and methodology used to compare the performance three different open-source MOM's by testing their messaging capabilities in publish/subscribe and point-to-point domains, functional comparison and qualitative study. All three MOM's support JMS API, and JMS API is used to develop the programs.

4.1 Maximum Sustainable Throughput (MST)

This is the point where the difference between the message-sending rate and receiving rate is zero. After this point the queue/topic will start to accumulate messages. At this point different messages rates are calculated. Publish rate is the rate at which a client can publish messages to the topic without any throttling in publish/subscribe domain. Subscribe rate is the rate at which the client can subscribe messages from the topic in publish/subscribe domain. Sending rate is the rate at which client can send messages to the queue in point to point messaging domain. Receiving rate is the rate at which client can receive messages in point-to-point messaging domain.

4.2 Latency

Latency in publish/subscribe is the time taken for all the messages to travel from the publisher program to the subscriber program. Latency in point to point is the time taken for all the messages to travel from the sending program to the receiving program. Latency is measured from time when the first message was published or sent to the queue up to time when the connection is closed on the publisher side or receiver side. The latency is measured by sending/publishing messages of different size (10, 512, 1024, 2048 KB) and the time is measured at the receiving/subscribing end.

4.3 Methodology

The programs were developed in Java using JMS API on the Windows Vista platform. The same program was used to measure the metrics for all the different messaging applications. The architectural details of the three applications are presented below. The performance tests were conducted with messages of varying size, which included 10Kbytes, 512Kbytes, 1024Kbytes, and 2048Kbytes.

The test programs accept runtime parameters to configure the key messaging factors that will be manipulated in the test run. This implementation conforms to the JMS 1.1 specification and will work with the three MOM's used in this study by changing the JNDI interface to a specific connection factory used by that MOM. Each invocation of the test runs in its own JVM, with all threads acting as Topic Publishers/Subscribers and Queue Senders/Receivers. Thus, it may be necessary to launch several instances of the test programs to cover the different message size.

4.4 Test Bed

The server and two wired clients were identical systems both in hardware and in software and were connected by an Ethernet switch. The details of the configuration were as follows: the Processor was an Intel ® Pentium ® 4 CPU 3.00 GHZ 2.99 GHz with 0.99 GB of RAM. The operating system used was Microsoft Windows Vista; and the network adapter was a Broadcom NetXtreme 57xx Gigabit Controller. In terms of connectivity, all three systems were connected to a Gigabit Ethernet Full duplex Ethernet Switch.

The software used on the server side was identical to the software used on the client side. Java: JDK 1.6 with Eclipse IDE, Active MQ 4.1, Mantaray MQ 2.0.1, and Open MQ 4.1.

5. Results

This chapter presents results obtained from this study on message rates, latency, statistical significance, functional comparison, and qualitative study from the perspective of a developer.

5.1 Publish Rate

Publish rate is the rate at which the messages can be published to the queue in publish/subscribe domain. Multithreading was used to depict multiple clients accessing the server at the same time.

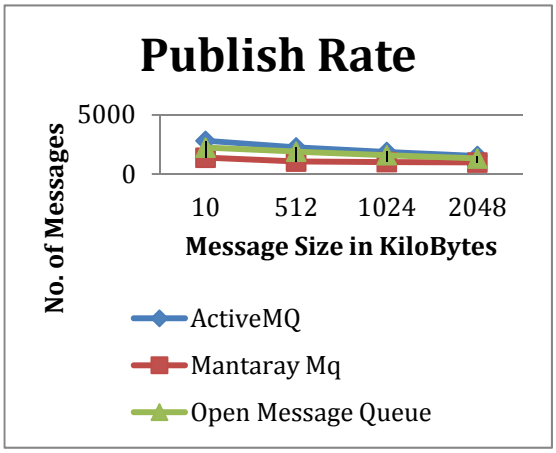

Figure 1. Publish rate

As shown in Figure 1 the x-axis represents the size of the message and the y-axis represents the number of messages. Figure 1 shows the decrease in performance as the message size increases. This is due to an increase

in assembling the message, transport time, processing time and header overhead. Active MQ performed better among the three products compared, while Mantaray MQ was the least efficient performer. The decrease in performance for Mantaray MQ was less compared to decrease in performance for Active MQ.

4.2 Subscribe Rate

Subscribe rate is the rate at which the messages can be subscribed from the queue. Multithreading was used to depict multiple clients accessing the server at the same time. As shown in Figure 2 the x-axis represents the size of the message and the y-axis represents the number of messages.

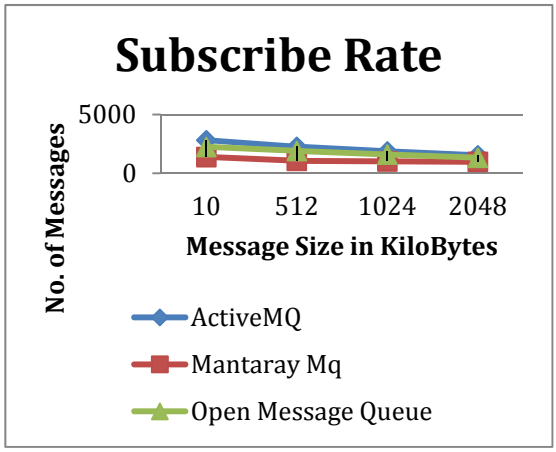

Figure 2. Subscribe rate

Figure 2 shows the subscribe rate: as the message size increases the performance decreases due to an increase in size of message to be retrieved, processed and removal of headers. Active MQ performed better among the three products compared, while Mantaray MQ was the least efficient performer. The difference in performance between Active MQ and Open MQ was not statistically significant.

4.3 Publish Vs. Subscribe

For the message sizes in which we tested Mantaray MQ, Open MQ and Active MQ, the publish rate and subscribe rate were similar, which is a factor to tell the MQ topics weren't filled up with messages and waiting to be subscribed. As the message size increased the performance decreased. As shown in Figures 3 to 5 the x-axis represents the size of the message and the y-axis represents the number of messages

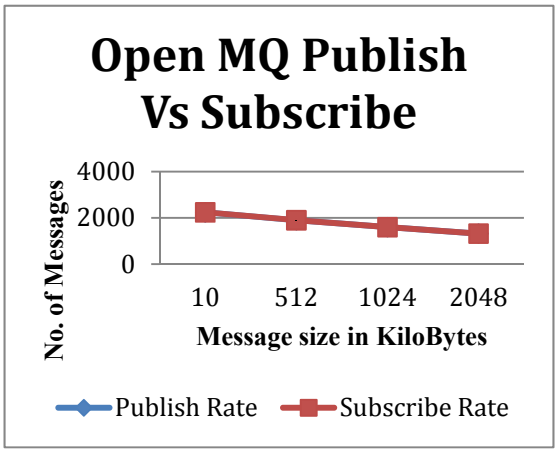

Figure 3. Open MQ publish Vs. subscribe rate

Figure 3 represents the performance comparison between the Open MQ publish and subscribe rate. The figure appears to show only one graph, but this is because the publish rate coincided with subscribe rate, and the two

graphs became indistinguishable.

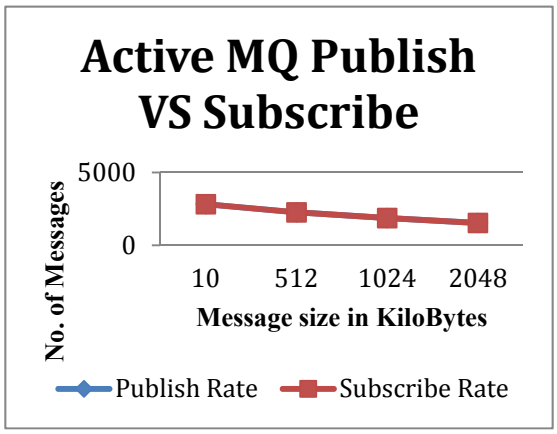

Figure 4. Active MQ publish Vs. subscribe rate

Figure 4 represents the performance comparison between the Active MQ publish and subscribe rate. The figure appears to show only one graph, but this is because the publish rate coincided with subscribe rate, and the two graphs became indistinguishable.

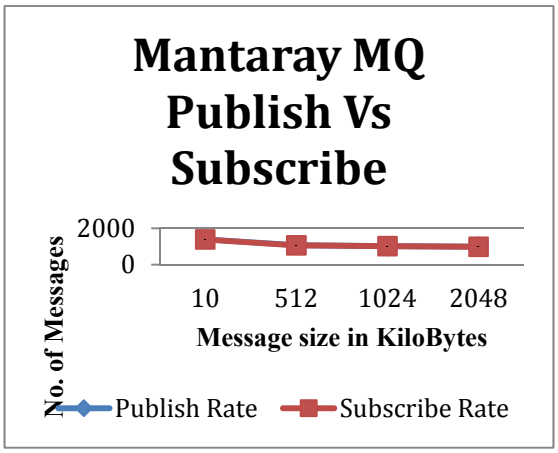

Figure 5. Mantaray MQ publish Vs. subscribe rate

Figure 5 represents the performance comparison between the Mantaray MQ publish and subscribe rate. The figure appears to show only one graph, but this is because the publish rate coincided with subscribe rate, and the two graphs became indistinguishable.

4.4 Sending Rate

Sending rate is the rate at which the messages can be sent to the queue in point-to-point domain. Multithreading was used to depict multiple clients accessing the server at the same time. As shown in Figure 10, the x-axis represents the size of the message and the y-axis represents the number of messages.

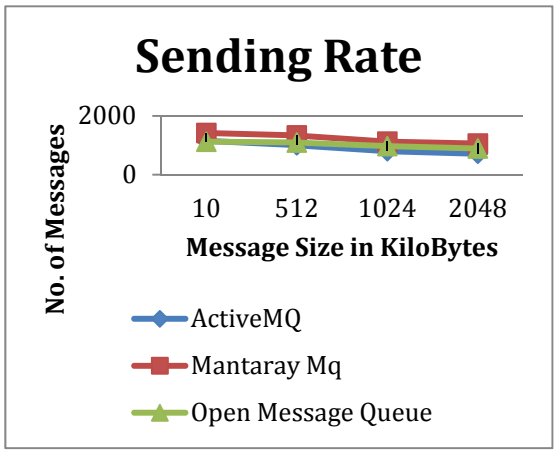

Figure 6. Sending rate

Figure 6 shows the sending rate of the three MQ providers, as the message size increases the through put decreases due to an increase in size of message to be sent to the queue, processed and header info. As opposed to publish/subscribe domain, in point-to-point messaging domain Mantaray MQ performed better. Active MQ performance decreased as the message size increased compared to Open MQ. At 10KB message size Active MQ performed well compared to Open MQ, but as the message size increased Open MQ performed better.

4.5 Receiving Rate

Receiving rate is the rate at which the messages can be read from the queue in point-to-point domain. Multithreading was used to depict multiple clients accessing the server at the same time. As shown in Figure 7, the x-axis represents the size of the message and the y-axis represents the number of messages.

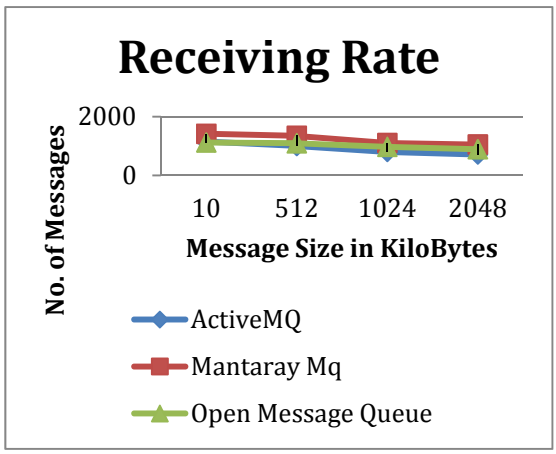

Figure 7. Receiving rate

Figure 7 shows the Receiving rate of the three MQ providers, as the message size increases the through put decreases due to an increase in size of message to be sent to the queue, processed and header info. Similar to sending rate Mantaray MQ performed better. Active MQ performance decreased as the message size increased compared to Open MQ. At 10KB message size Active MQ performed well compared to Open MQ, but as the message size increased Open MQ performed better. Active MQ was the least efficient performer of the three MQ's in Point to Point testing.

4.6 Sending Vs. Receiving

For the message sizes in which we tested Mantaray MQ, Open MQ and Active MQ, the sending rate and receiving rate were similar, which is a factor to tell the MQ's weren't queuing up with messages. As the message size increased the performance decreased.

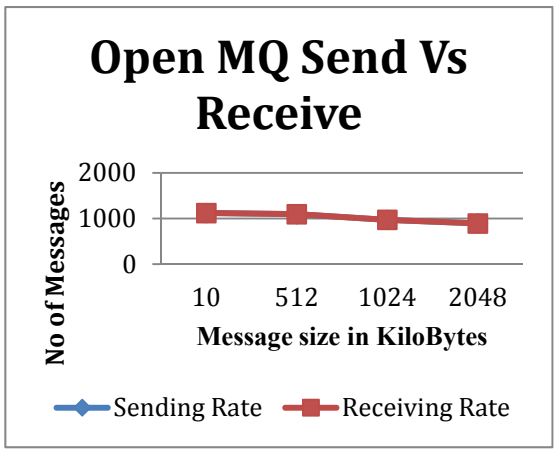

Figure 8. Open MQ Send Vs. Receive

Figure 8 represents the performance comparison between the Open MQ sending and receiving rate, The figure appears to show only one graph, but this is because the publish rate coincided with subscribe rate, and the two graphs became indistinguishable.

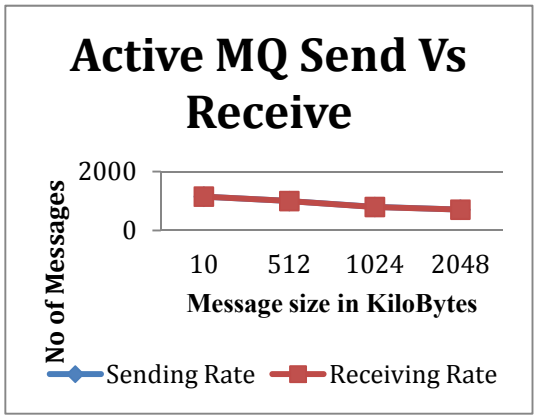

Figure 9. Active MQ Send Vs. Receive

Figure 9 represents the performance comparison between the Active MQ sending and receiving rate. The figure appears to show only one graph, but this is because the sending rate coincided with receiving rate, and the two graphs became indistinguishable.

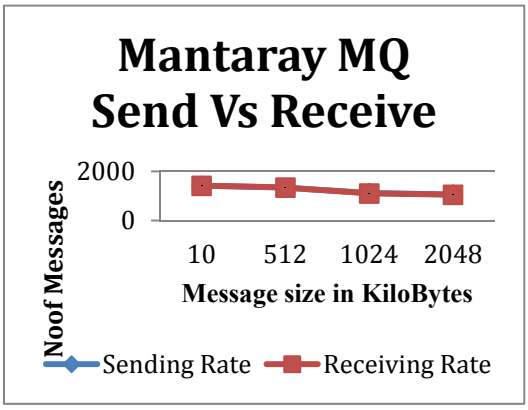

Figure 10. Mantaray MQ Send Vs. Receive

Figure 10 represents the performance comparison between the Mantaray MQ sending and receiving rate. The figure appears to show only one graph, but this is because the sending rate coincided with receiving rate, and the two graphs became indistinguishable. By looking at the Figure 10 it is clear that Mantaray MQ performed well during sending and receiving. As the message size increased, the number of messages decreased. Mantaray MQ was performing better in all message sizes in point-to-point messaging domain.

4.7 Latency

Latency in publish/subscribe is the time taken for all the messages to travel from the publisher program to the subscriber program. Latency in point to point is the time taken for all the messages to travel from the sending program to the receiving program. Latency is measured from time when the first message was published or sent to the queue up to time when the connection is closed on the publisher side or receiver side.

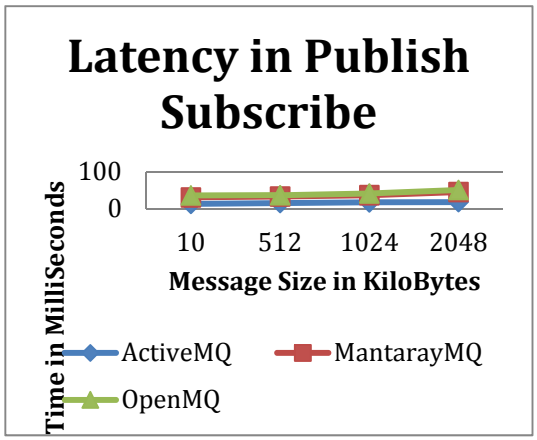

Figure 11. Latency in publish subscribe

From figure 11, we see that latency increased as the message size increased due to an increase in data that needs to be processed. Active MQ was the best among all three in Publish/Subscribe messaging. Open MQ had the highest latency among all three.

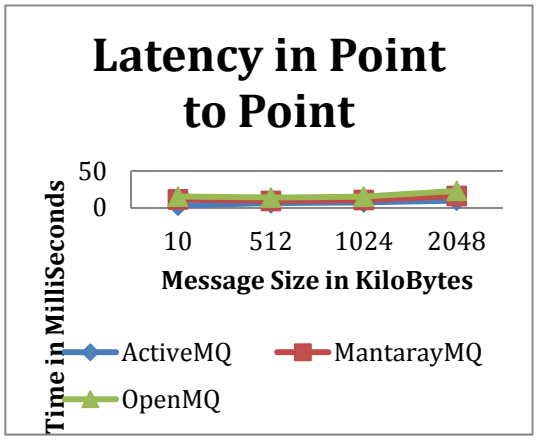

Figure 12. Latency in point-to-point

From the figure 12, latency increased as the message size increased due to an increase in data that needs to be processed. Active MQ was the best among all three in Point to Point messaging. Open MQ had the highest latency among all three.

4.8 Functional Comparison

Table 1 provides a functional comparison of the three products.

Table 1. Functional comparison

Function	Open MQ	Active MQ	Mantaray MQ
Underlying Messaging	Broker	Broker	Peer to Peer
Publish/Subscribe	Yes	Yes	Yes
Point to Point	Yes	Yes	Yes
Client API Supported	C Java	C C# Perl Python php Ruby Java	C++ Java
Administration	Console, CLI	Console CLI	Console(status only) Config File
Supported Frameworks	J2EE 1.4 JMS 1.1 JCA 1.5	J2EE 1.4 JMS 1.1 JCA 1.5	J2EE 1.4 JMS 1.1
Persistence	JDBC FILE	JDBC FILE	FILE
Scalability	Distributed Cluster	Distributed Cluster	N/A
Synchronous Messaging	Yes	Yes	Yes
Asynchronous Messaging	Yes	Yes	Yes
Operating System	Windows Linux Solaris	Windows Linux Solaris Mac	Windows Linux Solaris
JMX Support	Yes	Yes	Yes

4.9 Qualitative Study

Table 2 provides qualitative study of three products from a developer's perspective.

Table 2. Qualitative study

	Active MQ	Open MQ	Mantaray MQ
Ease with which the application was developed	Low	Low	High
Complexity of the middleware	High	High	Low
Architectural design	High	High	Low
Time required to develop the applications	Medium	Medium	Low
Security	High	High	Low
Ease of maintenance	High	High	Low
Documentation	High	Medium	Low
Support	High	High	Low

Mantaray MQ has the least setup required; all the required config files are generated during first run and can be modified later—Open MQ and Active MQ have significant install files and setup. The amount of documentation available made the job easy. Active MQ and Open MQ are fairly complex products offered with other features such as scalability and reliability. Architecturally, Open MQ and Active MQ are similar, but Mantaray MQ on the

other hand followed peer to peer architecture rather than client server architecture. Mantaray MQ documentation is not widely available and is tough to find support. Open MQ and Active MQ, on the other hand, do offer paid support if needed. Mantaray MQ is easily maintainable since it doesn't have much setup.

5. Conclusions and Future Work

The objective of this work was to benchmark three different open source MOM's by testing their messaging capabilities in both publish/subscribe and point-to-point domains. The performance benchmarking of Message Oriented Middleware was a difficult task. During the development phase, Mantaray MQ was found to be more difficult due to a lack of documentation and development tools. Open MQ and Active MQ, on the other hand, have good documentation and support for different IDEs and development tools. Active MQ performed better than the other two MQ providers in Publish/Subscribe messaging. Mantaray MQ performed better in Point to Point messaging but once the message size increased beyond 2048 Kbytes, Mantaray MQ couldn't handle the operation and crashed. Open MQ performed consistently with both Publish/Subscribe and Point-to-Point messaging. Mantaray MQ is good for small applications with fewer messages; such as small downloads, programs where no specific setup is required. Mantaray MQ is also recommended for applications that require configuration is done at runtime, and there is no dedicated server requirement. But Mantaray MQ is not suitable for enterprise applications, which require multiple features like clustering, high availability, scalability, and recovery from a crash. Also, all three products, when compared, provide support for a database in case of a broker or entire machine failure. From the latency tests it can be concluded that Open MQ performed better in both Publish/Subscribe and Point-to-Point. Mantaray MQ lacks features like scalability and reliability; on the other hand, Open MQ and Active MQ have good scalability and reliability features.

The results of this paper have opened other avenues of research that will need to be investigated further by asking why one MOM performs better than the other. Another avenue could be by considering other environment changes and other metrics, such as clustering, scalability and the performance impact caused by adding a database as persistent messaging. We could also further this work by studying the latest protocol AMQP—an improved version of MQ with more support by Java API.

References

Apache Software Foundation. (2013). *ActiveMQ Overview.* Retrieved July 3, 2013, from http://activemq.apache.org/code-overview.html

Bernstein P. (1996). Middleware, a Model for Distributed System Services. *Communications of the ACM, 39*(2), 86-98. http://dx.doi.org/10.1145/230798.230809

Brosey, W. D., Neal, R. E., & Marks, D. F. (2001). *Grand Challenges of Enterprise Integration* (pp. 221 -227). 8th IEEE International Conference on Emerging Technologies and Factory Automation, 2, 2, October 2001.

Chen, S., & Greenfield, P. (2004). *QoS Evaluation of JMS: An Empirical Approach* (pp. 5-8). Proceedings of the 37th Annual Hawaii International Conference on System Sciences.

Crimson Consulting Group. (2003). *High-Performance JMS Messaging: A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ*, Los Altos, CA, 2003.

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A. (2003). The Many Faces of Publish/Subscribe. *ACM Computing Surveys, 35*(2), 114–131. http://dx.doi.org/10.1145/857076.857078

IBM (2013). *WebSphere Software Main Page*. Retrieved July 3, 2013, from http://www-01.ibm.com/software/websphere/

Mantaray Download Page. (2013). Retrieved July 3, 2013, from http://sourceforge.net/projects/mantaray/

Oracle. (2013). *Oracle WebLogic Server Overview.* Retrieved July 3, 2013, from http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html

Pang, M., & Maheshwari, P. (2005), Benchmarking Message-Oriented Middleware – TIB/RV vs. SonicvMQ, Concurr. *Comput. : Pract. Exper., 17*(2), 15071526, 2005.

RedHat Linux. (2013). *JBoss Overview*. Retrieved from July 3, 2013, http://www.jboss.org/developer/tutorials.html

Sun Microsystems. (2013). *Open Message Queue Overview*. Retrieved July 3, 2013, from http://mq.java.net/overview.html

Tran, P., Greenfield, P., & Gorton, I. (2002). *Behavior and Performance of Message-Oriented Middleware Systems.* Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops,

2002.