Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware

Naveen Mupparaju
University of North Florida, n00487779@ospreys.unf.edu

PERFORMANCE EVALUATION AND COMPARISON OF DISTRIBUTED MESSAGING USING MESSAGE ORIENTED MIDDLEWARE

by

Naveen Mupparaju

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

August 2013

The thesis "Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware" submitted by Naveen Mupparaju in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee: Date

Dr. Sanjay P. Ahuja
Thesis Advisor and Committee Chairperson

Dr. Roger Eggen

Dr. Zornitza G. Prodanoff

Accepted for the School of Computing:

Dr. Asai Asaithambi
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

Dr. Mark A. Tumeo
Dean of the College

Accepted for the University:

Dr. Len Robertson
Dean of the Graduate School

ACKNOWLEDGEMENT

I wish to thank my parents and family for their encouragement and moral support to complete my thesis. I would like to thank Dr. Sanjay P. Ahuja for his guidance and constant support as my thesis advisor. I would also like to thank Dr. Roger Eggen and Dr. Zornitza Prodanoff for serving on my thesis committee and providing guidance and encouragement as well. Finally I wish to thank Dr. Karthikeyan Umapathy, Dr. Asai Asaithambi, and Mr. Jim Littleton for their valuable suggestions, which helped improve the writing of this thesis.

CONTENTS

List of Figures vii

List of Tables viii

Abstract ix

Chapter 1: Introduction 1

  1.1 Point to Point 2

  1.2 Publish/Subscribe 2

  1.3 Java Messaging Service 3

Chapter 2: Message Oriented Middleware 6

  2.1 Open Message Queue 6

  2.2 Active MQ 7

  2.3 Mantaray MQ 9

Chapter 3: Literature Survey 10

Chapter 4: Metrics and Research Methodology 13

  4.1 Maximum Sustainable Throughput 13

  4.2 Latency 14

  4.3 Methodology 14

  4.4 Test Bed 16

Chapter 5: Results 18

  5.1 Publish Rate 18

  5.2 Subscribe Rate 19

5.3 Publish Vs. Subscribe 20

5.4 Sending Rate 22

5.5 Receiving Rate 23

5.6 Sending Vs. Receiving 24

5.7 Latency 27

5.8 Statistical Significance 28

5.8.1 Latency Test 29

5.8.2 T-Test 30

5.9 Functional Comparison 32

5.10 Qualitative Study 33

Chapter 6 Conclusion and Future Work 34

References 36

Vita 38

LIST OF FIGURES

Figure 1: Point-to-Point Messaging. 2

Figure 2: Publish/Subscribe Messaging. 3

Figure 3: Open MQ Architecture. 7

Figure 4: Active MQ Architecture. 8

Figure 5: Publish Rate 18

Figure 6: Subscribe Rate 19

Figure 7: Open MQ Publish Vs. Subscribe rate 20

Figure 8: Active MQ Publish Vs. Subscribe Rate 21

Figure 9: Mantaray MQ Publish Vs. Subscribe Rate 22

Figure 10: Sending Rate 23

Figure 11: Receiving Rate 24

Figure 12: Open MQ Send Vs. Receive 25

Figure 13: Active MQ Send Vs. Receive 25

Figure 14: Mantaray MQ Send Vs. Receive 26

Figure 15: Latency in Publish Subscribe 27

Figure 16: Latency in Point-to-Point 28

LIST OF TABLES

Table 1: Latency Statistics. 29

Table 2: T-test of Message Rate. 30

Table 3: Functional Comparison. 32

Table 4: Qualitative Study.. 33

ABSTRACT

Message Oriented Middleware (MOM) is an enabling technology for modern event-driven applications that are typically based on publish/subscribe communication [Eugster03]. 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 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"[Brosey01]. 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, functional comparison and qualitative study from developers perspective.

Chapter 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 thesis 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.

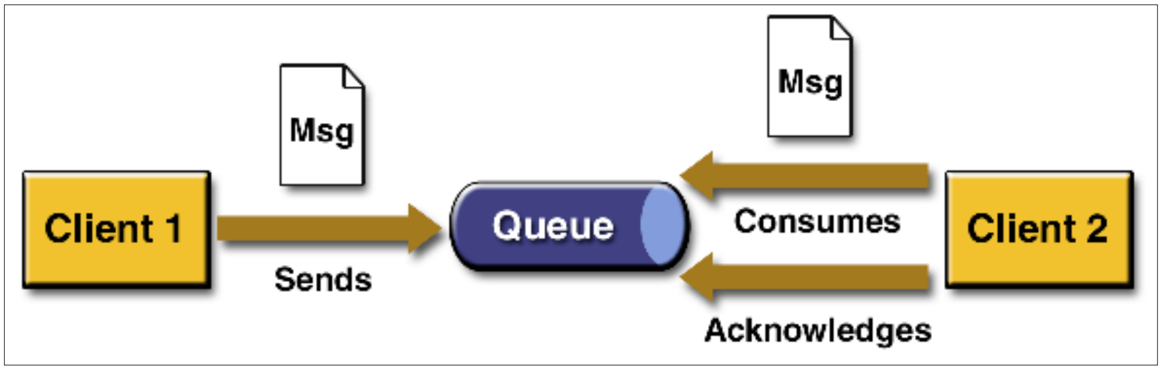

Figure 1: Point-to-Point Messaging [Sun13A].

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.

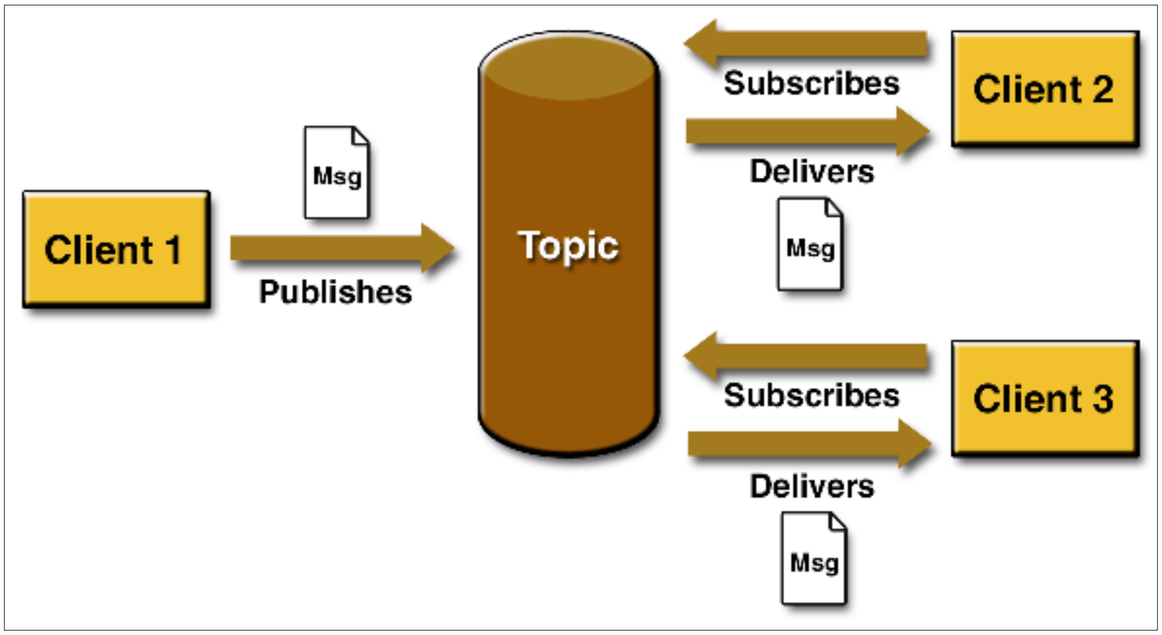

Figure 2: Publish/Subscribe Messaging [Sun13A].

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) [Sun13B]. 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 [Sun13A], Apache Active MQ [Apache13B], and Mantaray MQ [Mantaray13]. Brief descriptions of these are given in the chapter 2.

Chapter 2

MESSAGE ORIENTED MIDDLEWARE

This chapter 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. Figure 3 shows the architecture of Open MQ. 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.

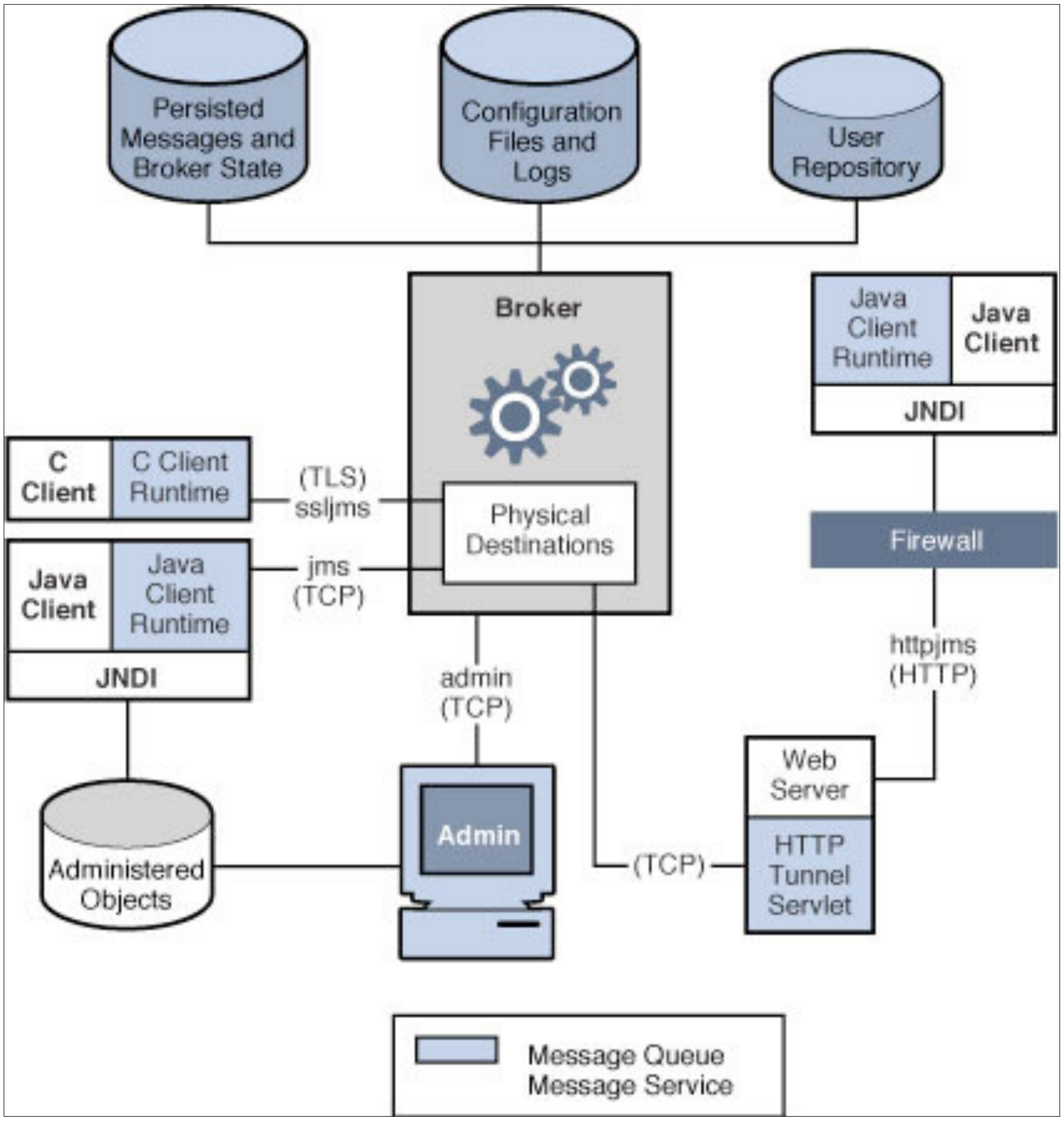

Figure 3: Open MQ Architecture [Apache02].

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. Figure 4 shows 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 [Apache13A], and can be installed on any operating system that supports Java.

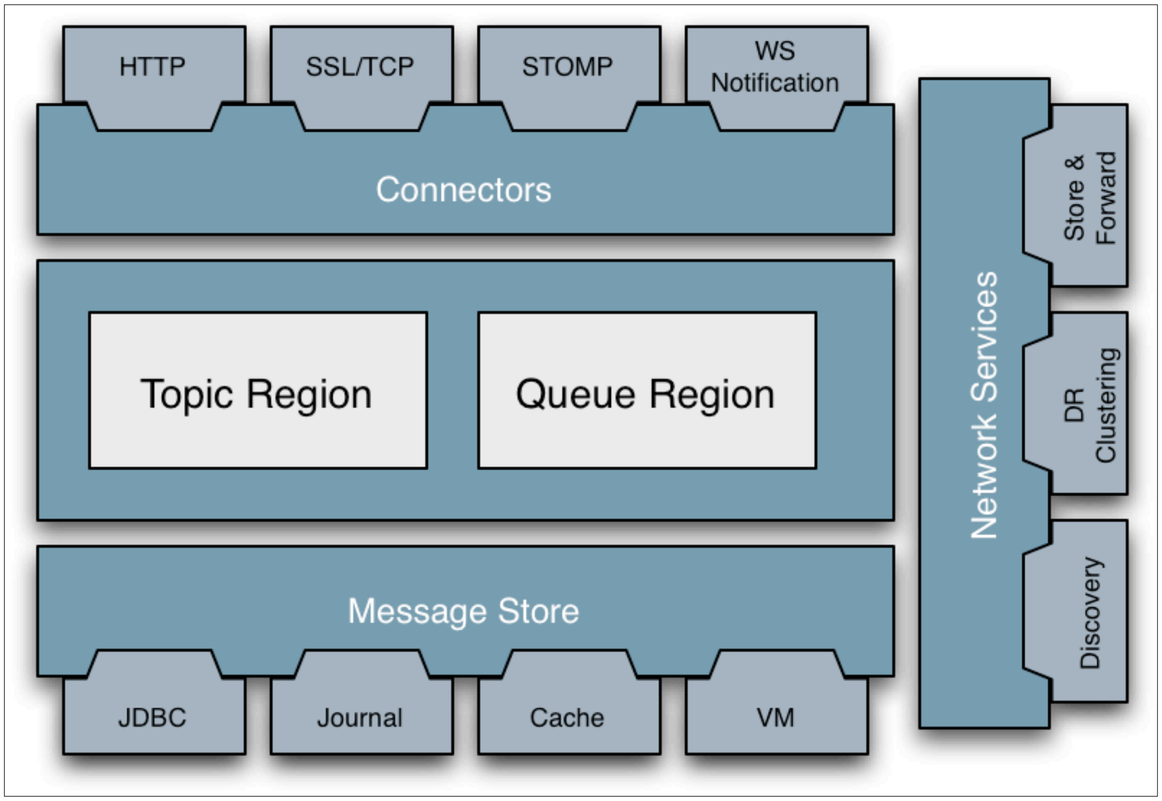

Figure 4: Active MQ Architecture.

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 [Redhat13], Weblogic [Oracle13] and Websphere [IBM13]. 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 config 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 config file. All peers that are involved in the communications should be configured in world map either statically or using Mantaray's Auto Discovery.

Chapter 3

LITERATURE SURVEY

Philip A. Bernstein, in “MIDDLEWARE: a Model for Distributed System Services” [Philip96] provides a discussion on different kinds of middleware, its evolution, and its services offered. The paper provides a good reference and also a model for different middleware applications, and explains in depth about the definition of middleware, what services should be offered in middleware and what services are not offered in middleware. The components that can be offered as part of middleware include: presentation management, computation, information management, communications, control, and system management. The paper also presents a good overview of framework, how it interacts with middleware, and provides an API for calling the middleware. In our thesis we use JMS API that is part of the J2EE framework.

Tran et al., [Tran02B] in a study titled, “Behavior and Performance of Message-Oriented Middleware Systems” provides a reference for behavior and performance of Message Oriented Middleware systems. The author also talks about how vendors—but not third parties—so far have published the metrics. The author of the paper explains the architecture of IBM MQ series, and how the performance was measured based on an open-loop test scenario with asynchronous senders and receivers.

Another research paper, “High-Performance JMS Messaging”**,** by the Crimson Consulting Group [Crimson03] was analyzed for this study. This report is a benchmark comparison of Sun Java System Message Queue 3.5 and IBM Websphere MQ 5.3. The paper gives an overview of both the technologies and discusses different testing variables. This reference is noteworthy because it compares an open source product, Sun Message Queue, with IBM MQ Series.

Chen et al., [Chen04] in a paper titled, “QoS Evaluation of JMS: an Empirical Approach”**,** discussed the relationship between JMS and MOM. The authors have presented different metrics for measuring the performance of message persistence, and compared two commercial applications.

P. Tran et al., [Tran02A] in, “J2EE Technology Performance Evaluation Methodology,” provided the reference for our evaluation approach—the benchmark application design and design principles of J2EE applications. Since this paper mainly discussed the J2EE evaluation standards we used design principles of the test application, such as identical hardware configuration, the same benchmark application for all products, and the same configuration products. The paper also mentioned different runtime considerations, which were used in this thesis mainly to turn off logging or any debug settings, and to stop the unnecessary process on the test machines. The paper also discussed different benchmark packages available for testing enterprise J2EE applications, which cannot be used directly for the thesis—although, the packages do serve as good references.

M. Pang et al., [Pang02] in “Benchmarking Message-Oriented Middleware – TIB/RV vs. Sonic MQ” is another reference used in this thesis which compares the performance between two commercial MOM’s TIB/RV and Sonic MQ. The paper includes an outstanding explanation of the architecture for both products and provides a good basis for a functional comparison between Message Oriented Middleware products discussed in Chapter 1.

As seen from the literature survey provided in this section, there exist studies, which have discussed and quantified the performance of commercial MOMs. The literature survey did not reveal any such studies pertaining to open source MOMs only; however, the existing papers were valuable in providing a general methodology and in suggesting metrics for this research.

Chapter 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.

Sample publish config file

```
     mode=pub
# **** Connection/Session info ****
     numconnections=1
     destination=ThesisTopic
     numdests=1
#  **** Producer options ****
     deliverymode=NONPERS
#  **** Message generation options ****
     msgtype=BYTE
     msgsize=4k
     msggenclass=BytesGenerator
     msgcache=1

#  **** Test timing and measurement ****
     sleepmsecs=0
     numrptintervals=10
     rptintervalsecs=10
     dowarmup=true
#  **** Connection and user specs ****
     jndiurl=none
     initialcontext=
org.apache.activemq.jndi.ActiveMQInitialContextFactory
     factoryname=connectionFactory
     url=vm://localhost
     user=Administrator
     password=Administrator
```

<u>mode</u>: The type of mode the sender/receiver is going to be. Different modes available are pub, sub, send and rec. Pub mode is used to publish messages to the topic, sub mode is used to subscribe messages from the topic, send mode is used to send messages to the queue, and rec mode is used to receive messages from the queue.

<u>numconnections</u>: number of connections from client to server

destination: the name of the topic/queue used in the testing.

deliverymode: this value specifies if the destination is persistent or not; different values accepted are NONPERS and PERS

msgtype: The type of JMS message used. The different types of JMS messages are text, bytes, stream, object, and map.

msgsize: size of the message to be sent.

msggenclass: the class used to generate different random messages for the purpose of testing.

msgcache: if the value is 1, the same message is used in entire test to avoid message generation delays

jndiurl: JNDI URL is used to instantiate the ConnectionFactory; if "none"; JNDI lookup is skipped.

initialcontext: JNDI Initial Context Factory class

url: url for server

factoryname: lookup name for the connection factory

username: username for connecting to the destination

password: password for connecting to the destination

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.

Chapter 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.

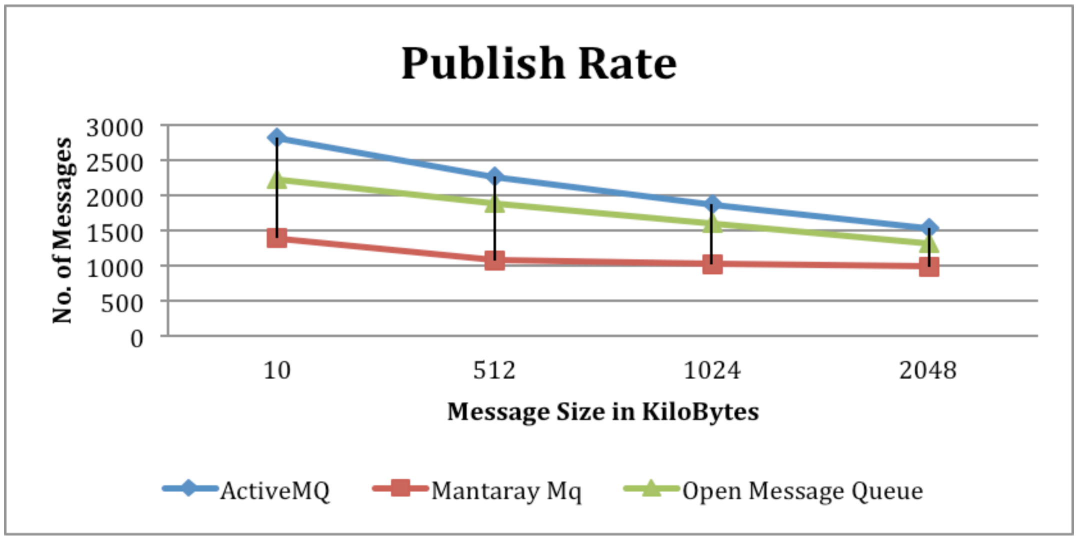

Figure 5: Publish Rate.

As shown in Figure 5 the x-axis represents the size of the message and the y-axis represents the number of messages. Figure 5 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.

5.2 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 6 the x-axis represents the size of the message and the y-axis represents the number of messages.

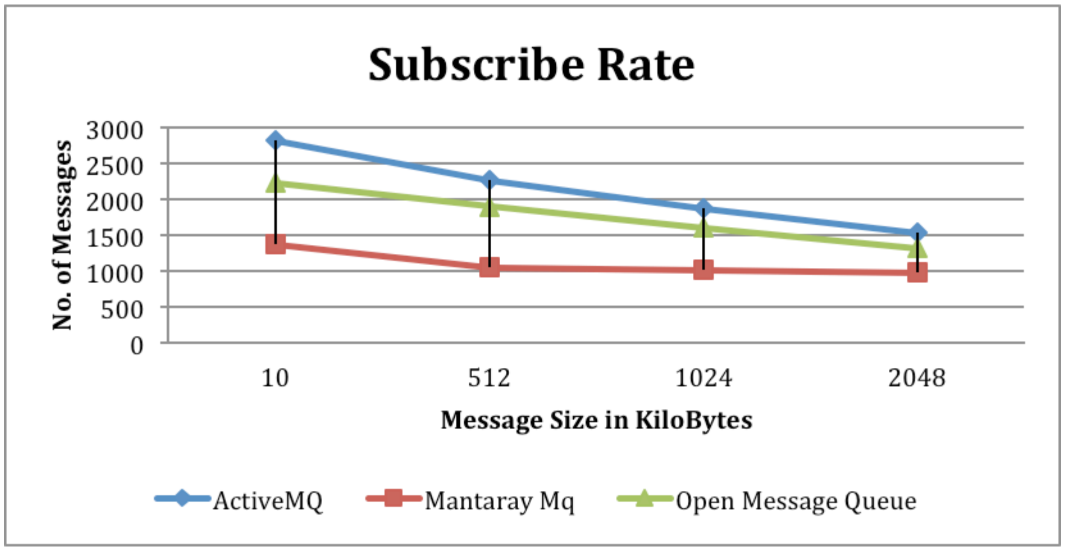

Figure 6: Subscribe Rate.

Figure 6 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.

5.3 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 7 to 9 the x-axis represents the size of the message and the y-axis represents the number of messages

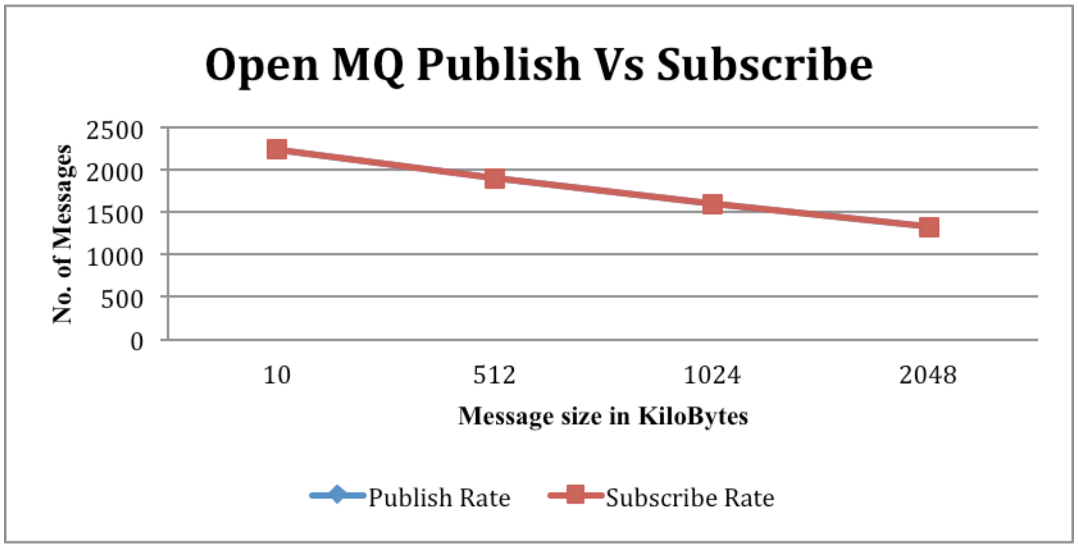

Figure 7: Open MQ Publish Vs. Subscribe Rate.

Figure 7 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.

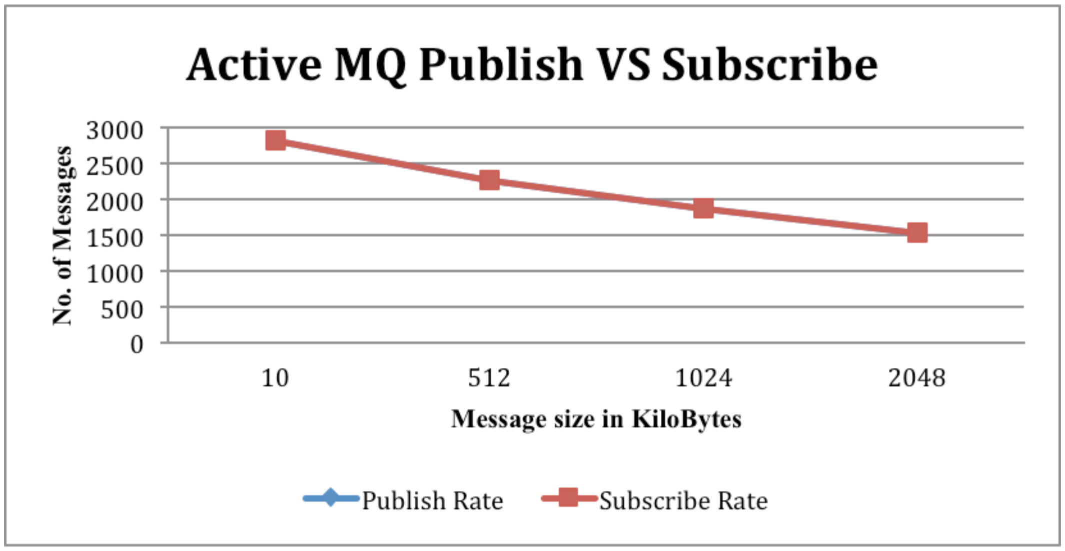

Figure 8: Active MQ Publish Vs. Subscribe Rate.

Figure 8 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.

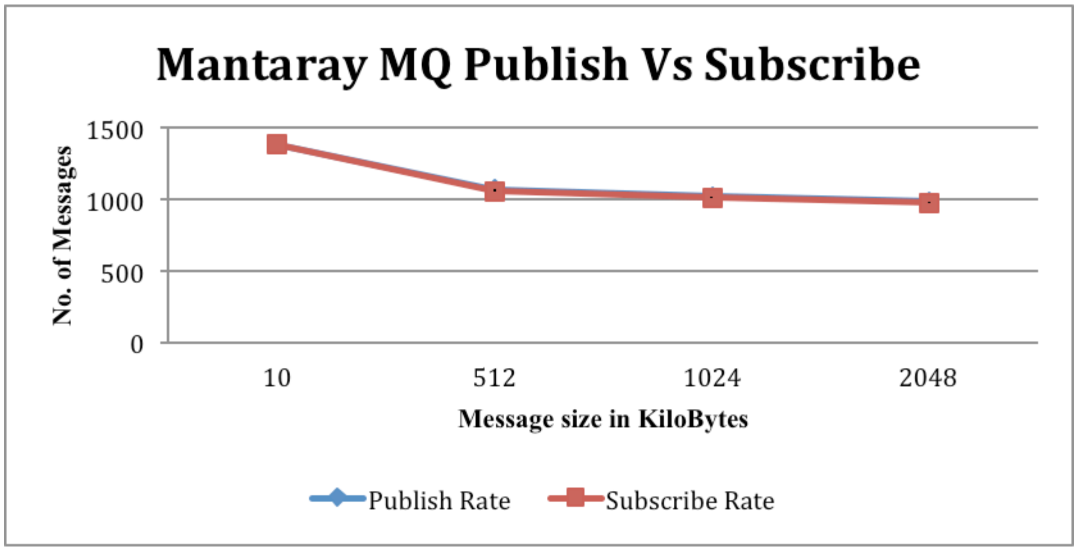

Figure 9: Mantaray MQ Publish Vs. Subscribe Rate.

Figure 9 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.

5.4 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.

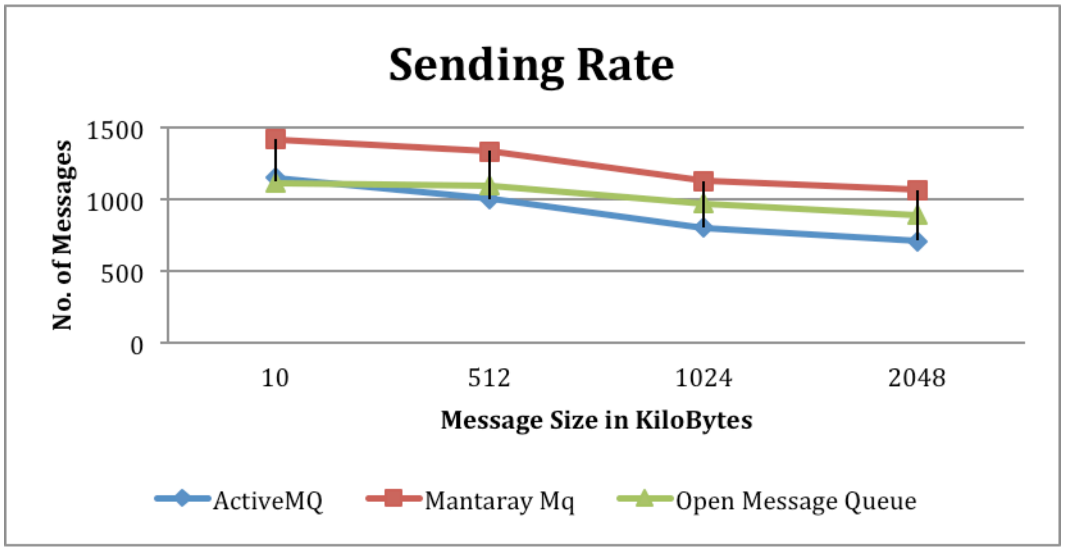

Figure 10: Sending Rate.

Figure 10 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.

5.5 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 11, the x-axis represents the size of the message and the y-axis represents the number of messages.

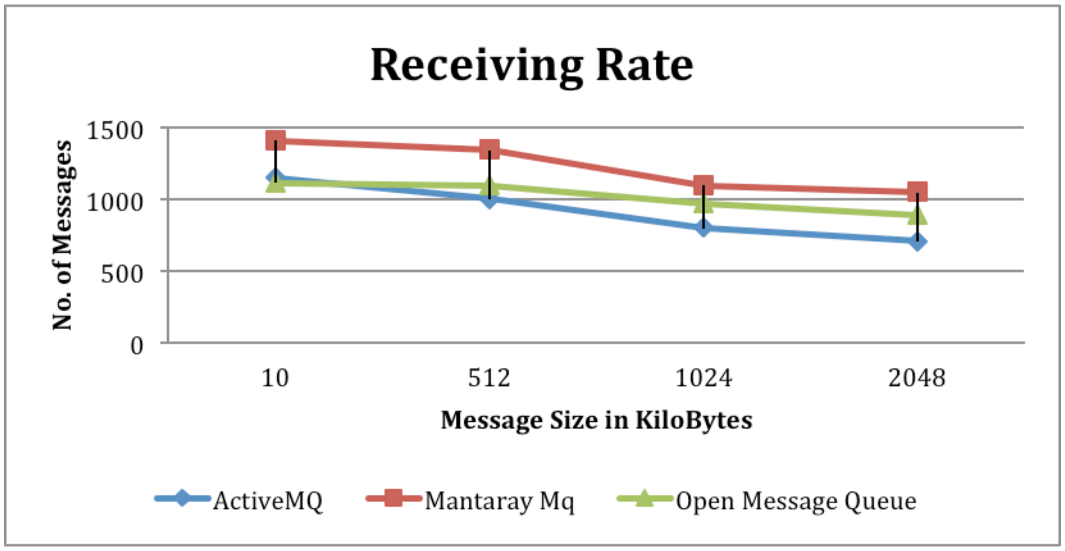

Figure 11: Receiving Rate.

Figure 11 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.

5.6 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.

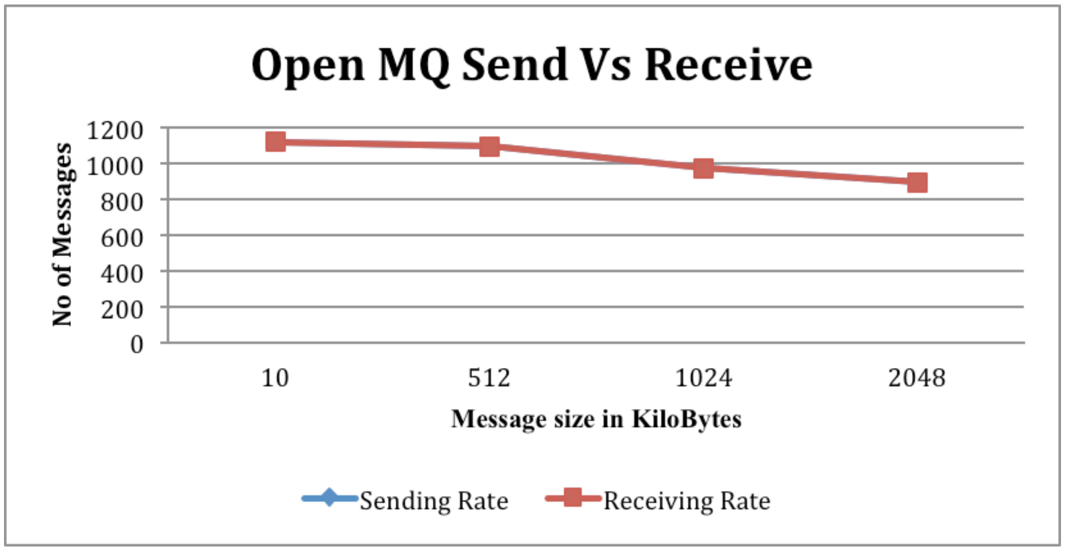

Figure 12: Open MQ Send Vs. Receive.

Figure 12 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.

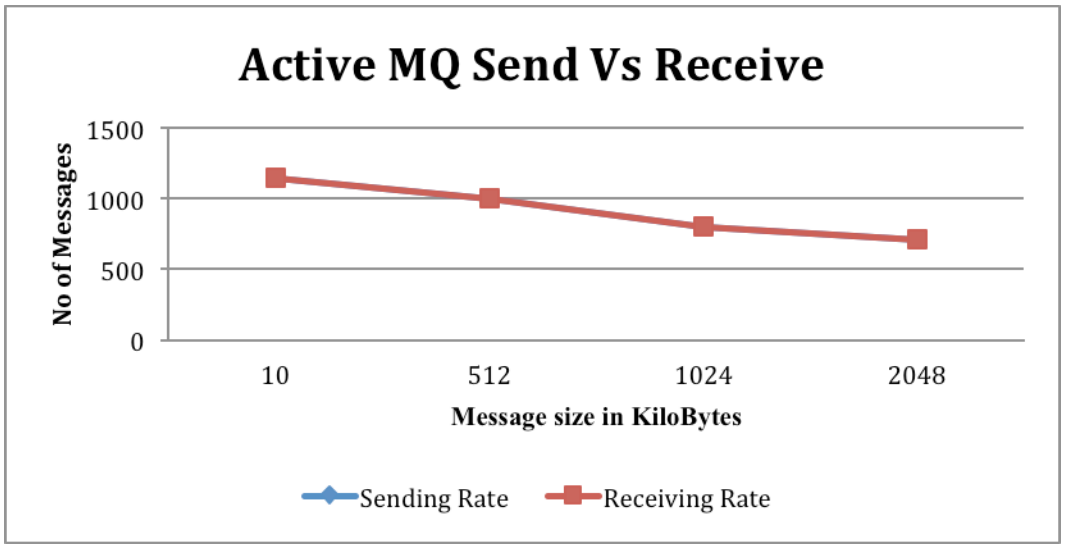

Figure 13: Active MQ Send Vs. Receive.

Figure 13 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.

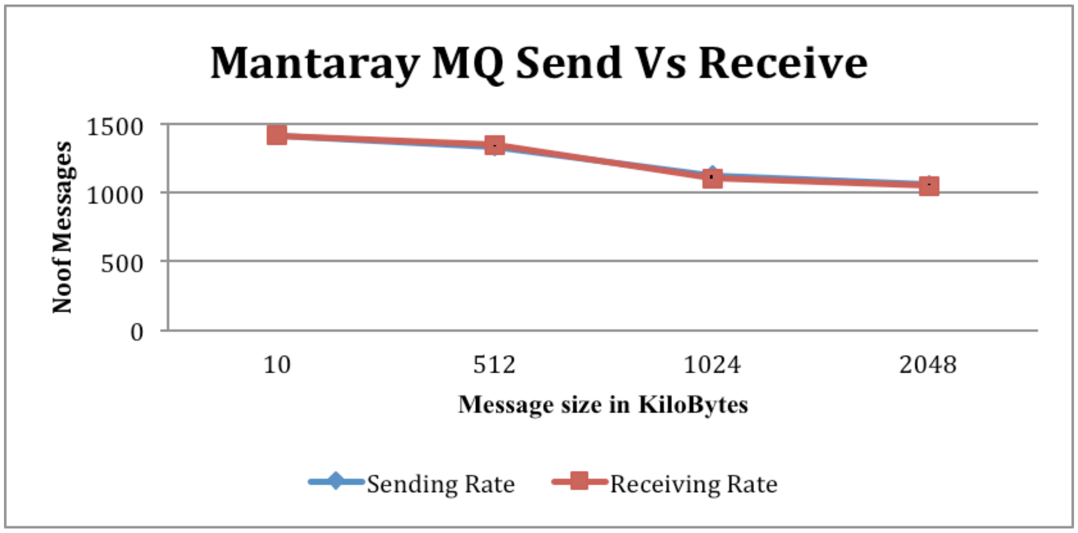

Figure 14: Mantaray MQ Send Vs. Receive.

Figure 14 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 14 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.

5.7 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.

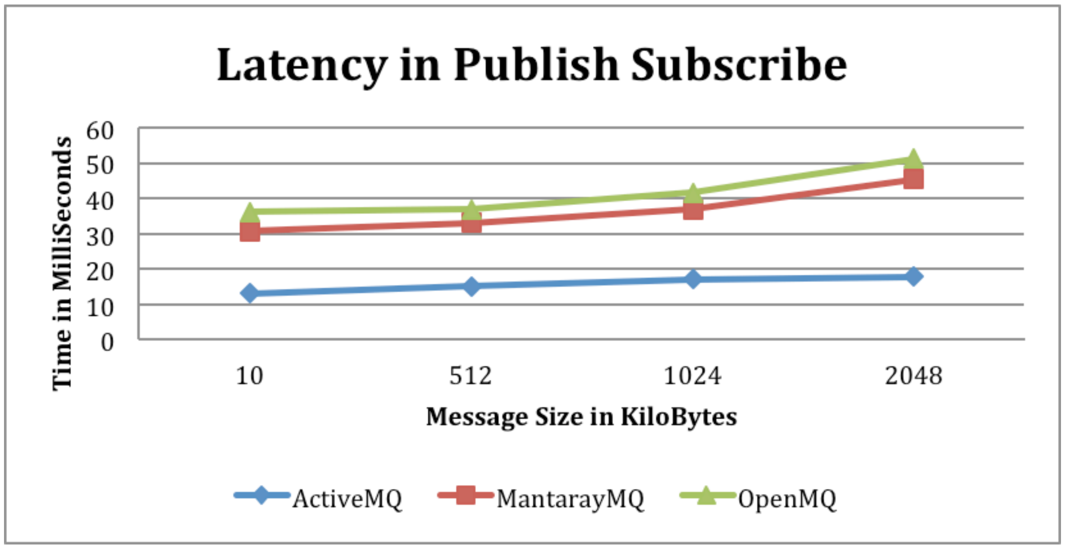

Figure 15: Latency in Publish Subscribe.

From the figure 15, 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.

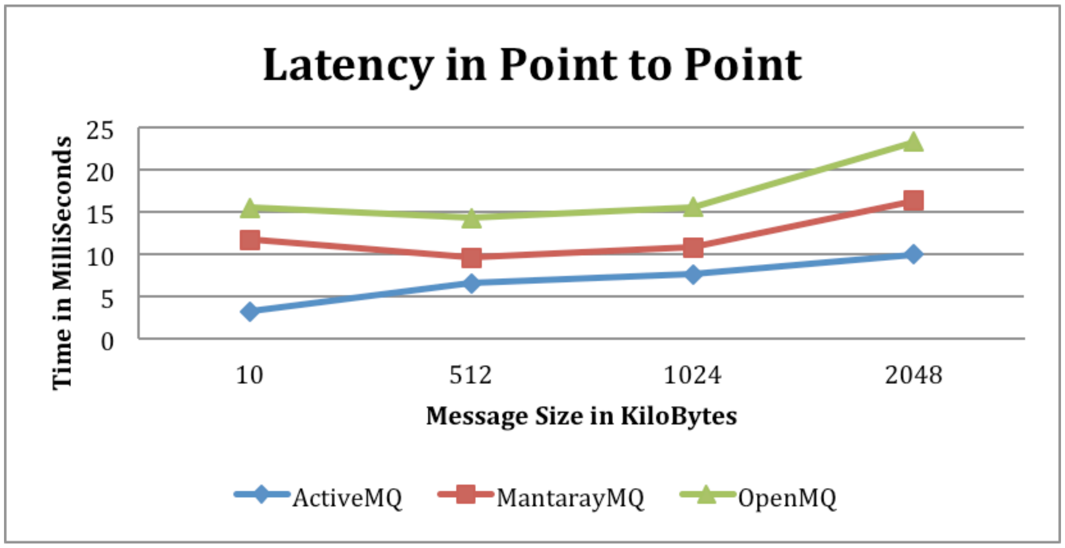

Figure 16: Latency in Point-to-Point.

From the figure 27, 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.

5.8 Statistical Significance

Tests for statistical significance tell us whether there is the probability of relationship between two variables or if they are merely random variables. In other words, statistical significance means that there is a good chance that we are right in finding a relationship between two findings [Walonick97]. An experiment is considered valid if the p value is less than 0.05. Since a t-test is limited to two variables, we used ANOVA to find the statistical significance, which can be applied to more than two variables. In this case, these variables refer to the latency related result set obtained for each of the three MOMs: Active MQ, Open MQ, and Mantaray MQ.

5.8.1. Latency Test

Table 1 shows the results of an ANOVA test applied on latency metrics. In Table 1:

- df shows the degrees of freedom that relates to the number of values in the data;
- SS denotes the sum of squares (sum of the squared differences of each score from the mean of all scores);
- MS denotes the mean square (estimates of variance and are computed by dividing the sum of squares by the degrees of freedom);
- F denotes the F ratio computed by dividing the MS for between groups (different values between different MQ's) by the MS for within groups (different values within same MQ);
- P-values if the probability of obtaining an F as large or larger than one computed in the data assuming the null hypothesis is true and F_{crit} is the highest value of F that can be obtained without rejecting the null hypothesis.

The p-value of 0.034 is obtained. Since this p-value < 0.05, the results of this test are considered to be statistically significant.

Source of Variation	Sum of Squares	df	Mean Square	F	F_{crit}
Between Groups	18.304	2	9.152	0.034	4.256
Within Groups	16.490	9	1.832		
Total	34.794	11			

Table 1: Latency Statistics.

5.8.2 T-Test

In order to determine the statistical significance of the result sets obtained for the messaging rates of Open MQ and Active MQ, we conducted a T-Test between their result sets. Specifically, the t-test was used to determine whether the mean of a population (Open MQ) differed significantly from the mean of another population (Active MQ). To determine whether the difference is statistically significant, the t-test calculates a p-value. The p-value is the probability of the differences in the data occurring from the population by chance and is usually set at 0.05 by statisticians and this conveys a significance level. In this case, a p-value < 0.05 indicates that the difference in the result sets is statistically significant. The t-test was conducted using degrees of freedom (*df*) = 6. The more data we have, the smaller the sampling error is likely to be. The degrees of freedom value takes this into account when calculating the p-value.
 Likewise, the t-test was conducted for the result sets of Active MQ and Mantaray MQ, and for the result sets of Mantaray MQ and Open MQ as shown in Table 2.

Test	Active MQ vs. Mantaray MQ		Active MQ vs. Open MQ		Open MQ vs. Mantaray MQ	
	t stat	P(T≤t) one-tail	t stat	P(T≤t) one-tail	t stat	P(T≤t) one-tail
Publish	3.42	0.007	1.04	0.167	−2.95	0.012
Subscribe	3.94	0.003	1.04	0.167	−3.61	0.005
Sending	−1.98	0.047	−0.92	0.194	1.56	0.043
Receiving	−1.95	0.049	−0.92	0.194	1.52	0.048

Table 2: T-test of Message Rate.

For publish rate—by comparing the three products separately the performance difference between Active MQ and Open MQ was not statistically significant (p=0.167). But Active

MQ (p=0.007) demonstrated a significant difference in performance over Mantaray MQ and Mantaray MQ (p=0.012) demonstrated a significant difference in performance over Open MQ.

For subscribe rate—by comparing the three products separately the performance difference between Active MQ and Open MQ (p=0.167) was not statistically significant. But Active MQ (p=0.003) demonstrated a significant difference in performance over Mantaray MQ and Mantaray MQ (p=0.005) demonstrated a significant difference in performance over Open MQ.

For Sending rate— by comparing the three products separately the performance difference between Active MQ and Open MQ (p=0.194) is not statistically significant. But Mantaray MQ (p=0.047) demonstrated a significant difference in performance over Active MQ and Mantaray MQ (p=0.043) demonstrated a significant difference in performance over Open MQ.

For Receiving rate—— by comparing the three products separately the performance difference between Active MQ and Open MQ (p=0.194) is not statistically significant. But Mantaray MQ (p=0.049) demonstrated a significant difference in performance over Active MQ and Mantaray MQ (0.048) demonstrated a significant difference in performance over Open MQ.

The difference in performance between Open MQ and Active MQ was not statistically significant because of the similar architecture but different implementation. Mantaray MQ has a completely different peer-to-peer architecture discussed in chapter 2.

5.9 Functional Comparison

Table 3 provides the functional comparison of the three products.

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

Table 3: Functional comparison.

5.10 Qualitative study

Table 4 provides qualitative study of three products from a developer's perspective.

	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

Table 4: Qualitative Study.

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.

Chapter 6

CONCLUSION 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 thesis 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

Print Publications:

[Brosey01]
Brosey, W.D., R.E. Neal and D.F. Marks, “Grand Challenges of Enterprise Integration”, 8th IEEE International Conference on Emerging Technologies and Factory Automation, 2, 2, (October 2001), pp. 221 and 227.

[Chen04]
Chen, S. and P. Greenfield, “QoS Evaluation of JMS: An Empirical Approach”, Proceedings of the 37th Annual Hawaii International Conference on System Sciences, (January 2004), pp. 5-8.

[Crimson03]
Crimson Consulting Group, “High-Performance JMS Messaging: A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ”, Los Altos, CA, 2003.

[Eugster03]
Eugster, P.T., P.A. Felber, R. Guerraoui and A. Kermarrec, “The Many Faces of Publish/Subscribe”, ACM Computing Surveys, 35, 2 (June 2003), pp. 114–131.

[Pang02]
Pang, M. and P. Maheshwari, “Benchmarking Message-Oriented Middleware – TIB/RV vs. SonicvMQ”, 2002.

[Philip96]
Bernstein P., “Middleware, a Model for Distributed System Services”, Communications of the ACM, 39, 2, (February 1996), pp. 86-98.

[Tran02A]
Ran, S., D. Palmer, P. Brebner, S. Chen, I. Gorton, J. Gosper, L. Hu, A. Liu and P. Tran, “J2EE Technology Performance Evaluation Methodology”, CSIRO Mathematical and Information Sciences, Australia, 2002.

[Tran02B]
Tran, P., P. Greenfield and I. Gorton,” Behavior and Performance of Message-Oriented Middleware Systems”, Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops, (2002), pp. 645.

Electronic Sources:

[Apache13A]
Apache Software Foundation, ActiveMQ Download,
http://activemq.apache.org/index.html, last accessed July 3, 2013.

[Apache13B]
Apache Software Foundation, ActiveMQ Overview, http://activemq.apache.org/code-overview.html, last accessed July 3, 2013.

[IBM13]
IBM, WebSphere Software Main Page, http://www-01.ibm.com/software/websphere/, last accessed July 3, 2013.

[Mantaray13]
Mantaray Download Page, http://sourceforge.net/projects/mantaray/, last accessed July 3, 2013.

[Redhat13]
RedHat Linux, JBoss Overview, http://www.jboss.org/developer/tutorials.html, last accessed July 3, 2013.

[Sun13A]
Sun Microsystems, Open Message Queue Overview, http://mq.java.net/overview.html, last accessed July 3, 2013.

[Sun13B]
Sun Microsystems, Java Message Service Tutorial,
http://docs.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/basics.html#1023671, last accessed July 3, 2013.

[Oracle13]
Oracle, Oracle WebLogic Server Overview,
http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html, last accessed July 3, 2013.

[Walonick97]
Walonick, D.S., Survival Statistics, Bloomington, MN (1997),
http://www.statpac.com/surveys/surveys.pdf, last accessed July 3, 2013.

VITA

Naveen Mupparaju has a Bachelor of Technology degree in Computer Science from Jawaharlal Nehru Technological University, 2006 and expects to receive a Master of Science in Computer Science from the University of North Florida, August 2013. Dr. Sanjay P. Ahuja of the University of North Florida is serving as Naveen's thesis advisor. Naveen is currently employed as an IT Senior Consultant at Oracle America, Inc., and has been with the company for 3 years. Prior to that, Naveen worked 2 years as a System Administrator with IxReveal, Inc., Jacksonville.

Naveen has on-going interests in Cloud Computing, Application Servers, Message Queues, real-time parallel systems, Mobile Computing, iOS, Android and Linux, and has extensive experience with Linux, Apache, Weblogic, OBIEE, Windows Administration, tomcat and Networking. Naveen has extensive programming experience in C, Java, and C#. Naveen's academic work has included use of Java, html, Visual BASIC, .net, data mining, and SQL as well. Naveen is fluent in English, Telugu and Hindi.