Benchmarking Message-Oriented Middleware – TIB/RV vs. SonicMQ

Michael Pang[♣] and Piyush Maheshwari
School of Computer Science and Engineering
The University of New South Wales
Sydney NSW 2052, Australia
e-Mail: piyush@cse.unsw.edu.au

Abstract

Message-oriented middleware (MOM) has become a vital part of the Enterprise Application Integration (EAI) projects. This is because it can be used to pass data and workflow in the form of messages between different enterprise applications. The performance of EAI greatly depends on how effectively the MOM performs. This paper presents a benchmark comparison between two industry well-known MOMs – Tibco Rendezvous (TIB/RV) and Progress SonicMQ. Although the two MOMs are very similar in certain aspects, their native implementation and architecture are very different. We provide an unbiased benchmark reference to the middleware selection process. The primary objective of our work was to evaluate the MOMs by testing their effectiveness in the delivery of messages in publish/subscribe and point-to-point message domains, their program stability and the system resource utilization.

1. Introduction

1.1. What is MOM?

MOM is a specific class of middleware that supports the exchange of general-purpose messages in a distributed application environment (Figure 1). MOM systems ensure message delivery by using reliable queues and by providing directory, security, and administrative services required to support messaging.

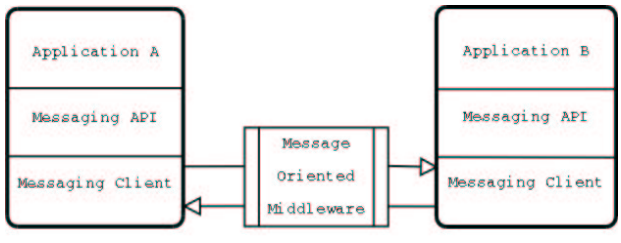


Figure 1: Message-Oriented Middleware

By sending the messages asynchronously, the sender of a MOM message does not have to wait for an acknowledgement from the receiver. It can simply send the message and continue with other processes. Rather than connecting directly to a remote objects, the program passes the message via connection to the MOM which then sends the message to the receiving application.

1.2. Two Messaging Models

1.2.1. Publish and Subscribe <pub/sub> (One-to-Many)

The sender is called the *publisher* and the receiver is called the *subscriber*. One producer can publish a message to many consumers through a virtual channel called a *topic*. Consumers can choose to subscribe to a topic they are interested in. Any messages addressed to a topic are delivered to all the subscriber of that topic.

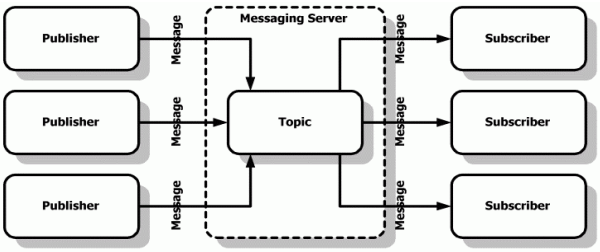


Figure 2: Publish/Subscribe Model [1]

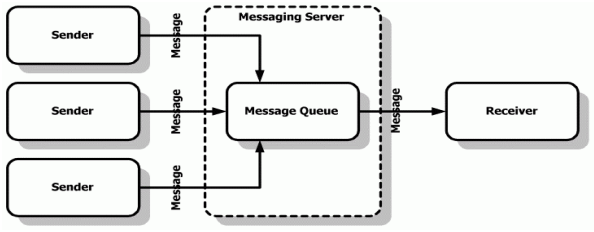


Figure 3: Point-to-Point Model [1]

[♣] Now working with SAP Australia. E-mail: michael.pang@sap.com

Every consumer receives a copy of the message being published. This is called a *push-based* model, where messages are automatically broadcast to consumers without them having to request or poll the topic for new messages. A published message is received by all interested parties, and parties can subscribe and unsubscribe at will. Publish/subscribe works similarly to signing up for an email distribution list.

1.2.2. Point-to-Point <PTP> (One-to-One)

In the point-to-point arrangement, there is typically a single sender and a single receiver. This can be done either synchronously or asynchronously via a virtual channel called a message queue.

In this paper, we compare industry well-known MOMs – Tibco Rendezvous (TIB/RV) and Progress SonicMQ. Although the two MOMs are very similar in certain aspects, their native implementation and architecture are very different. This paper provides an unbiased benchmark reference to the middleware selection process. The primary objective of our work was to evaluate the MOMs by testing their effectiveness in the delivery of messages in publish/subscribe and point-to-point message domains, their program stability and the system resource utilization.

The rest of the paper is organised as follows. Section 2 discusses the two MOMs presented in the paper. Section 3 discusses the benchmarking issues and system tuning. Section 4 gives a brief description on the tests we have implemented for the benchmarking purpose. Finally, Section 5 presents the results obtained from the tests and also provides a detailed analysis.

2. Benchmarking the MOMs

This section introduces the two MOMs, TIBCO Rendezvous and Progress SonicMQ.

2.1. TIBCO Rendezvous (TIB/RV)

TIBCO has been one of the leading providers of EAI since its establishment 20 years ago and TIB/RV is one of the most widely used messaging middleware in enterprises.

2.1.1. Architecture

TIB/RV is based on a distributed architecture [2]. An installation of TIB/RV resides on each host on the network. Hence it eliminates the bottlenecks and single points of failures could be handled. It allows programs to send messages in a reliable, certified and transactional manner, depending on the requirements. Messaging can be delivered in point-to-point or publish/subscribe, synchronously or asynchronously, locally delivered or sent via WAN or the Internet. Rendezvous messages are self-describing and platform independent. Figure 4 below represents the main architecture of TIB/RV.

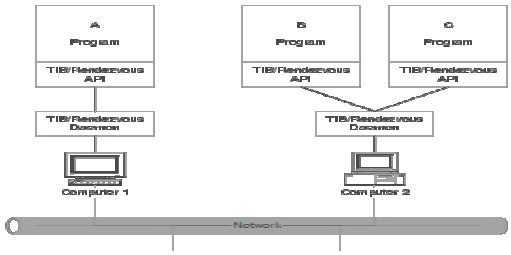


Figure 4: TIB/Rendezvous Operating Environment

2.1.2. Components

TIB/RV is composed of three main components:

- *RV Daemon* (RVD) — responsible for the delivery of messages within a LAN.
- *RV Agent* (RVA)
- *RV Routing Daemon* (RVRD)

2.1.3. RV Daemon (RVD)

Figure 4 shows the architecture of RV in a LAN environment. Notice how the RV Programs A, B and C, connect to RVD through TIB/RV API, and the RVD is connected to the network. RVD listens to the network for every message intended for the programs.

The RV daemon arranges the details of data transport, packet ordering, receipt acknowledgment, retransmission requests, and dispatching information to the correct program processes. The daemon hides all these details from TIB/Rendezvous programs. RV programs create the message users want to send, and passes it onto RVD, which is then responsible for passing them to the destination(s).

RVD is a background process that sits in between the RV program and the network. It is responsible for the delivery and the acquisition of messages, either in point-to-point or publish/subscribe message domain. It is the most important component of the whole TIB/RV.

Theoretically, there is an installation of RVD on every host on the network. However, it is possible to connect to a remote daemon, which sacrifices performance and transparencies.

2.1.4. How does a message get to its destination?

Publish/Subscribe (One-to-Many)

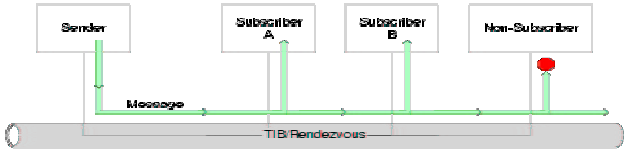


Figure 5: Reliable Multicast Message [3]

Figure 5 shows how messages are publish/subscribe in a LAN environment with the use of RVD. The RV Sender program passes the message and destination topic to RVD. RVD then broadcasts this message using User Data Packet (UDP) to the entire network. All subscribing computers with RVDs on the network will receive this message. RVD will filter the messages which non-subscribers will not be notified of the message. Therefore only subscriber programs to the particular topic will get the messages.

Point-to-Point (One-to-One)

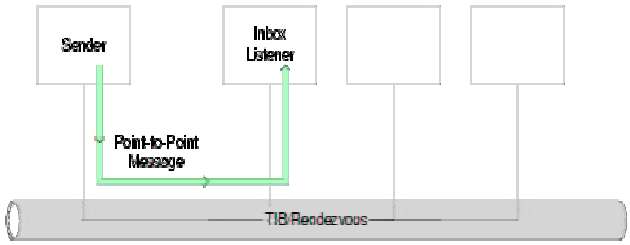


Figure 6: Point-to-Point Messaging in TIB/RV

In TIB/RV, point-to-point messages are fairly similar to publish/subscribe, only in a one-to-one model instead of one-to-many. The receiver creates a unique "inbox name" that establishes an address for receiving point-to-point messages. To send a point-to-point message, the sending program must know the inbox name of the destination. The receiver makes its inbox name known by multicasting it to potential senders using a prearranged subject name.

Once the sending program receives the recipient's inbox name, it will broadcast the message with inbox name as the topic to the network using UDP. The recipient's RVD will be able to receive the message when it realised the topic is it's own unique inbox name.

2.2. Progress SonicMQ

SonicMQ is one of the newer MOMs in the market. It claims to be the first JMS implementation, and has outstanding performances competitive with existing MOM technologies, such as IBM MQSeries. SonicMQ is written in 100% pure Java, supports XML messaging, and HTTP tunnelling to allow SonicMQ to work over the Internet.

The underlying mechanism of SonicMQ is its "broker" that facilitates the movement of messages across the network. It is a Java executable that requires Java Virtual Machine (JVM) to run.

The communication protocols that can be used with SonicMQ include TCP, HTTP and SSL. Since it uses common Internet protocol, SonicMQ can extend its deployment to the Internet. It also provides *bridges* to many other popular MOMs that allow messages to be sent and received between SonicMQ and other MOMs.

2.2.1. Architecture

There are three types of configurations a user can choose from:

- *Single-broker Configuration*

  Under this configuration, there is one broker which is being shared across a few nodes.

- *Multi-broker Clusters*
- *Multi-node Configurations*

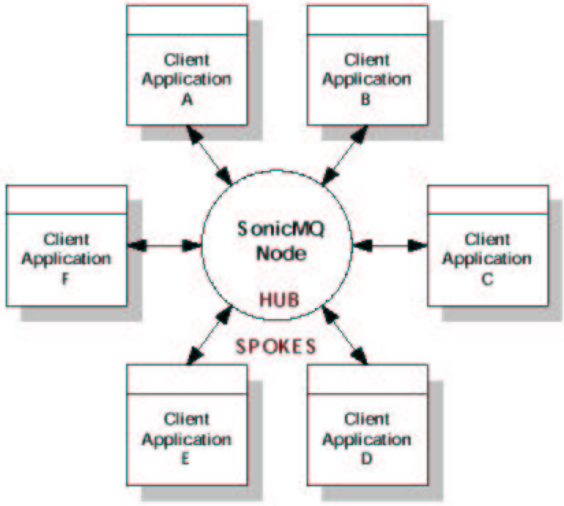


Figure 7: SonicMQ Single Broker Hub-Spoke Model

Figure 7 shows a single broker configuration. The broker is the most important underlying implementation of SonicMQ. It is responsible for delivering and acquiring of messages within a LAN environment. It is a client-server model, where many clients connect to a single broker. The connection can be via TCP (for LAN), SSL (for security encryption), or even HTTP (to connect to external entities).

The downside with single broker configuration is that scalability is limited by the capabilities of the node machine. Also the system is dependent on the single broker machine (node), hence leading to a bottleneck of the system at the node. The whole system may collapse if the node goes down. To solve this problem a multi-broker cluster must be used.

2.2.2. SonicMQ Performance

There exist a benchmark report for SonicMQ by Progress Software [5]. SonicMQ showed outstanding performances compared to IBM MQSeries and Fiorano FioranoMQ, (both are JMS implementations) under WinNT platform.

2.3. TIB/RV vs SonicMQ Functional Summary

The functional features for TIB/RV and SonicMQ are listed in the table below for comparison:

	TIB/RV	SonicMQ
Underlying Messaging mechanism	RVD	Broker
Publish/Subscribe	yes	yes
Point-to-Point	yes	yes
Subject-based Addressing	yes	no
Location Transparency	yes	no
Synchronous Messaging	yes	yes
Asynchronous Messaging	yes	yes
Guaranteed Messaging	Certified Messaging	Durable Messaging
Fault Tolerance	yes	yes
WAN/Internet Support	RVA/RVRD	Dynamic Routing Architecture
Persistent Messaging	Ledger Storage	Persistent

3. Benchmarking

In order to find out how well a distributed messaging infrastructure performs under heavy load with a number of concurrent connections, a benchmark test must be undertaken. This section discusses the issues concerned with planning and deployment of such a complex task.

3.1. Aims of this Benchmark Report

The aim of the benchmark tests performed is to determine the following:

1. The effectiveness of the MOMs – messages per second (MPS)
2. The time taken for a batch of messages to be delivered
3. The effects of increasing the number of publisher and subscriber connections in a pub/sub scenario
4. The effects of increasing the number of sender and receiver pairs in a PTP scenario
5. The effects of constant publisher/sender as opposed to a period publisher/sender
6. The memory and CPU utilization of the MOMs

These objectives are achieved by:

- Writing some Java programs using the APIs provided by the MOMs
- Using a Linux system utility "top" to monitor memory and CPU utilization of different processes in the system.

3.2. Benchmark Environment

The following is the environment utilized to perform the benchmark:

1x Server:	*1x Client:*
Intel Pentium III 450 processor 196Mb SDRAM 13G IBM IDE Hard disk	AMD K6-850 Processor 256Mb SDRAM 30G IBM ATA Hard disk

3.2.1. Network Connection

Network Interface: 10BaseT NE2000 compatible network card

Cable: Twist Pair

Length of cable: Each 1-meter.

Hub: RJ45 5 Port Hub

3.2.2. Operating System

Mandrake Linux 8.1 – 2.4.8-2 version Kernel

4. Test Description

This section describes the Java programs we have implemented for testing the two MOMs. The programs for TIB/RV use TIBCO's own Java API, and the programs for SonicMQ use JMS API developed by Sun Microsystems. As the programs are developed with different APIs, we had to make special effort to ensure the programs for the same test are as similar as possible to yield comparable results. Each test aims to evaluate different issues relating to MOMs.

Different scenarios are constructed to evaluate the MOMs. Tests 1 and 2 assume a constant producer of messages i.e. the MOMs are always busy throughout the entire test duration. We measure the pub/sub rates and PTP send/receive rates which represent the behaviour of the MOMs to such environment.

As discussed previously, different MOM programs have to make different procedures to start up and connect to the MOM. Thus, Test 3 measures the connection time of the subscribers. Finally, Test 4 attempts to measure the memory and CPU utilization of the two MOMs.

Each of these tests was tested with different input parameters, e.g., changing message size, number of message producers and consumers, etc.

4.1. Test 1 – Publish/Subscribe Fixed and Test 2 – Point-to-Point Fixed

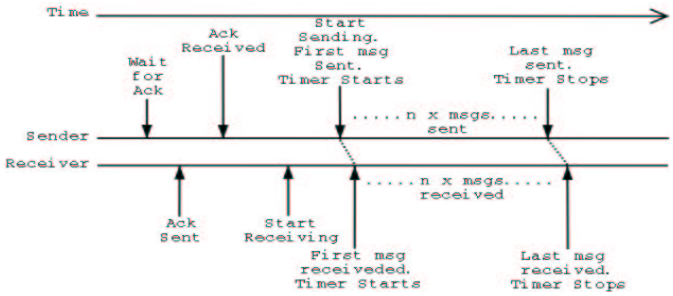
Tests 1 and 2 aim to measure the publish/subscribe and the point to point messaging rate of the MOMs.

Message Domain: Publish/Subscribe and Point to Point

Description: The sender and receiver programs measure the rate to publish and subscribe a fixed number of

messages, with specified message size, number of publishers and subscribers. The programs print the average rate after sending all the messages.

Program Flow:



Input Variables: The programs are configurable using a properties file (configuration file) by changing:

1) Number of publishers/senders

2) Number of subscribers/receivers

3) Number of messages to publish/send

4) Size of the message being publish/sent.

5) Broker/service group, and daemon the programs connect to.

4.2. Test 3 – Publish/Subscribe – Connection Time

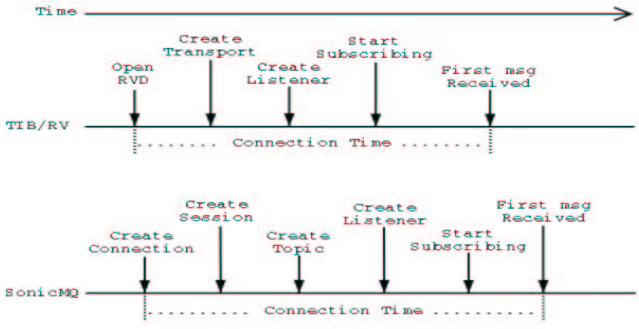
Test 3 aims to find out how quickly a subscriber can start the RVD/Sonic broker and start receiving messages.

Message Domain: Publish/Subscribe

Details: The publisher keeps on publishing messages to the topic "Publish". The subscriber records the starting time, and records the time of the first message being received.

Before the subscriber starts, the publisher is already publishing. Hence we can ensure our measurement will not be affected by the late arrival of the messages.

Program Flow:



Input Variables: The total amount of messages to publish must be inputted as an argument to run the publisher.

4.3. Test 4 – Memory and CPU Utilization

Test 4 aims to measure the memory and CPU utilization of each MOM.

"Top" is a Linux system utility that monitors the CPU and Memory consumption of different active processes. Since the utilization changes every few second, we can only compare by finding the average of the data. As such we wrote a Unix shell script that captures the data from the output of the "top" utility every second for the duration of the program.

5. Test Results and Analysis

This section presents all test results achieved from the tests described in the previous section. A detailed analysis is provided to evaluate the results.

5.1. Part 1 – Results

5.1.1. Test 1 – Publish/Subscribe Fixed

Test 1-A: Varying the number of subscribers

Fixing number of publishers, varying number of subscribers.

Number of publisher(s) = 1
Number of subscribers varies from 1 to 9
Message size = 10 kilobytes
Number of messages sent = 2000
Total data published by each publisher= 20Mb

Total data subscribed = Number of Publishers x Total data published (20Mb)

Average publish rate is the rate of the messages that can be passed from the publisher program to the MOM. Avg Subscribe Rate = total messages received by all subscribers / total time to receive all messages.

Total Duration is the time taken for all the messages to travel from the publisher program to the subscriber program. It is measured from time when the first message was published up to time when the connection is closed on the publisher side.

Publisher Transfer Rate = Total Messages / Total Duration

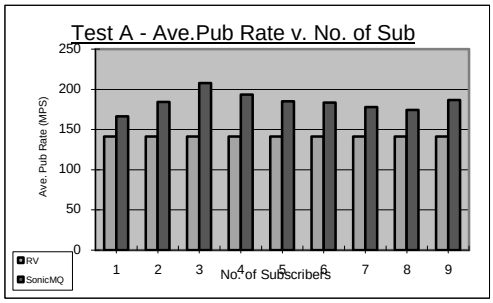


Figure 8: Average Publish Rate v/s Number of Subscribers

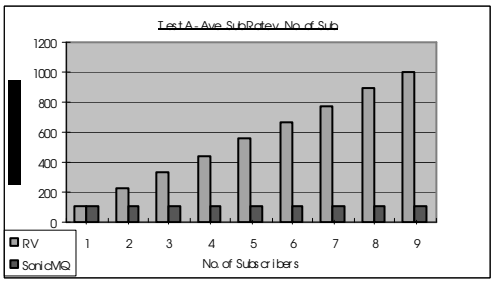


Figure 9: Average Subscribe Rate v/s Number of Subscribers

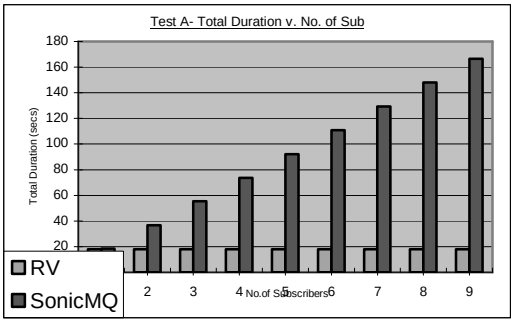


Figure 10: Total Duration v/s Number of Subscribers

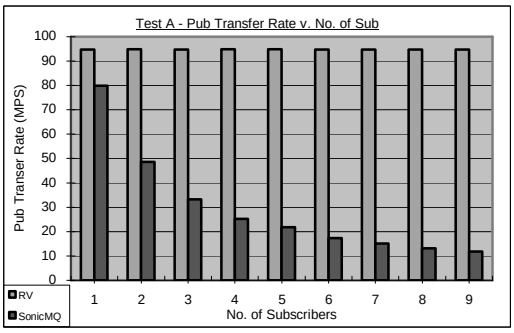


Figure 11: Publish Transfer Rate v/s Number of Subscribers

Test 1-B: Varying the number of publishers

Fixing number of subscribers, varying number of publishers.

Number of Subscriber(s) = 1 while Number of Publishers varies from 1 to 9

Message size = 10 kilobytes

Number of messages sent = 2000

Total data published by each publisher= 20Mb

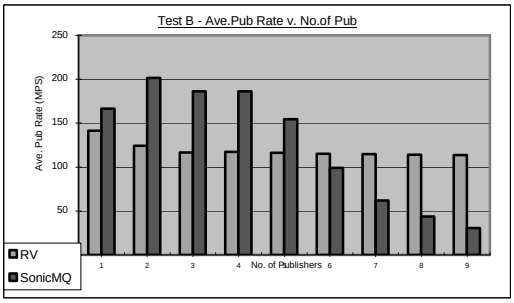


Figure 12: Average Publish Rate v/s Number of Publishers

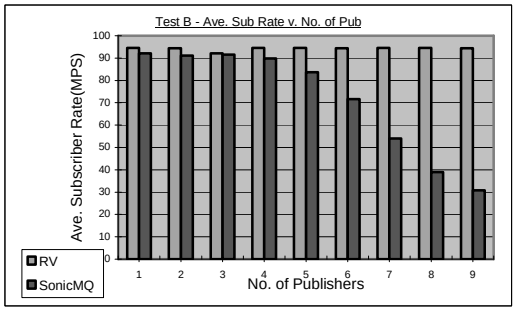


Figure 13: Average Subscribe Rate v/s Number of Publishers

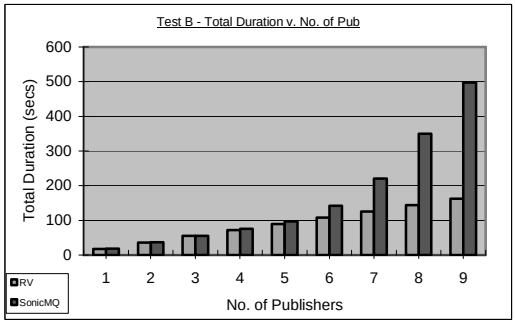


Figure 14: Total Duration v/s Number of Publishers

Figure 15: Publish Transfer Rate v/s Number of Publishers

5.1.2. Test 2 – Point-to-Point Fixed

Fixed number of messages

Number of sender/receiver pairs varies from 1 to 5

Message size = 10 kilobytes

Number of messages sent = 2000

Total data sent by each sender = 20Mb

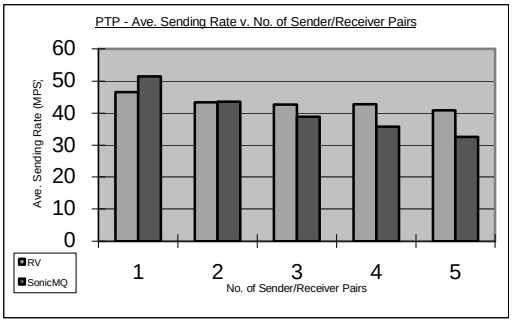


Figure 16: Average Sending Rate vs No of Sender/Receiver Pairs

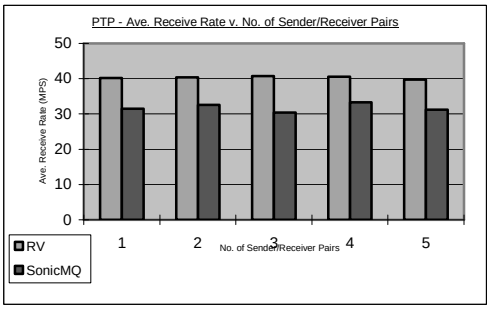


Figure 17: Average Receive Rate vs No. of Sender/Receiver Pairs

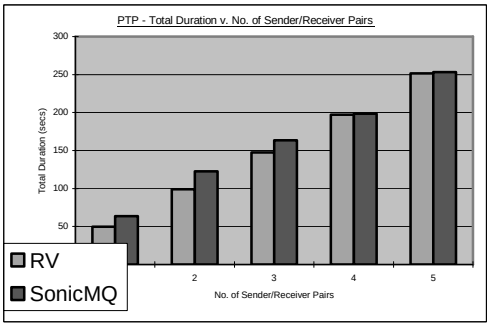


Figure 18: Total Duration vs No of Sender/Receiver Pairs

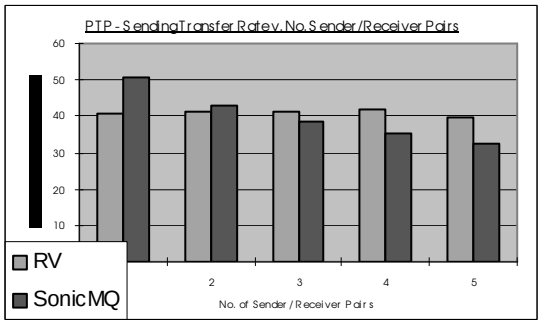


Figure 19: Sending Transfer Rate vs No. of Sender/Receiver Pairs

Average sending rate is the rate of the messages that can be passed from the sender program to the MOM. Average receive rate is the rate of the messages that can be passed from the MOM to the subscriber program.

Total Duration is the time taken for all the messages to travel from the sender program to the receiver program. It is measured from time when the first message was sent up to time when the connection is closed on the sender side. The sender will not be able to close the connection until all the messages reach the receiver.

Sending Transfer Rate is the rate that messages are transferred from the sender program to the receiver program. Sending Transfer Rate = Total Messages / Total Duration

5.1.3. Test 3 – Publish/Subscribe – Connection Time:

Table 1: Connection Time Comparison

TIB/RV	0.56 secs
SonicMQ	2.55 secs

5.1.4. Test 4 – Memory, CPU Utilization:

Table 2: Memory and CPU Utilization Comparison

	TIB/RV		SonicMQ	
	Memory Utilization (MB)	CPU Utilization	Memory Utilization (MB)	CPU Utilization
RVD/Broker Start up	2%	10-20%	60.0	60-85%
Pub/Sub – Publisher	55%	60%	50-70%	70-80%
Pub/Sub - Subscriber	8%	5%	25%	20%
PTP – Sender	50%	60%	50-70%	75%
PTP – Receiver	5%	5%	20%	15%

5.2. Part 2 – Results Analysis

The following section analyzes the previous results.

5.2.1. Publish/Subscribe (Test 1)

It was obvious that TIB/RV performed well in all the tests. It shows an impressively stable publish/subscribe rates. On the other hand, SonicMQ suffers a scalability issue, showing decreasing performance as the number of publishers or number of subscribers increase.

Tests 1A and 1B give an in-depth comparison between the two MOMs.

Test 1A – Varying the Number of Subscribers

In this test, there is only one publisher, and the number of subscribers gradually increases from 1 to 9. (The more subscribers, more messages will be subscribed).

For Figure 8, SonicMQ beats TIB/RV in the average publishing rate. However, it shows the rate for TIB/RV is much stable and consistent compared to that of SonicMQ. All tests results obtained were done 3-5 times to minimize the error that might have been involved. Should SonicMQ

be more consistent, all the bars in this graph should be of same height.

TIB/RV has an average publishing rate of 141.26MPS, where SonicMQ has an average 184.42MPS. Hence SonicMQ is approximately 30% faster than TIB/RV for the rate to pass the messages from the publisher program to the MOM.

Figure 9 shows the relationship between the average subscribing rate and the number of subscribers. SonicMQ shows a constant average receiving rate of 108.4MPS, regardless of changes in the number of subscribers. TIB/RV on the other hand, shows a direct relationship to the increase of number of subscribers. The more subscribers results in the higher receiver rate.

One of the reasons is because the total duration for the same amount of messages published does not change regardless of the number of subscribers for TIB/RV (Figure 10).

The RVD in the publisher just broadcasts the message out to everyone inside the network using the UDP protocol. Hence regardless of the number of subscribers, the total duration does not change. The SonicMQ broker requires a separate TCP connection to each of the subscriber. Hence the more subscribers there are, the longer it takes for SonicMQ to deliver the same amount of messages.

Figure 11 illustrates the changes in the average publishing rate relative to the number of subscribers. TIB/RV shows a constant publishing rate, unaffected by the changes in the number of subscribers. As explained above the total duration for SonicMQ increases as the number of subscribers, hence according to the formula provided in how the publish rate is calculated, it explains why the publish rate decreases as more subscribers are introduced.

Test 1B – Varying the Number of Publishers

This test demonstrates the effect of the changes in number of publishers with a single subscriber.

In Figure 12, the average publishing rate illustrates the rate the messages passed from the publishers to the MOM. When the number of publishers is less than 6, SonicMQ is a lot faster. However, when the number of publishers gets higher, SonicMQ's performance drops rapidly. As it can be seen, the average publishing rate for SonicMQ is very inconsistent. However 3-5 repeats of each test was performed to minimize the error of the results. The constant average publishing rate of TIB/RV is 116 MPS.

Figure 13 shows the relationship between the average subscribing rate to the number of publishers. As there are more publishers, TIB/RV subscriber has no effect to changes. However for SonicMQ, it shows a gradual decline, as there are more publishers and more messages being sent.

Figure 14 shows the total duration to publish/subscribe all the messages specified. This is measured from the moment the messages leave the publisher program in the server, until the time the connection is closed. The connection will not close until the subscriber program in the client receives all the messages. The figure shows that the more messages to be published, the longer it takes, as there are more processing time involved. It also shows that for the same amount of messages and publishers, SonicMQ takes longer to complete than TIB/RV. The difference increases as there are more publishers and messages to send. This is accounted for due to the difference in the underneath architecture and implementations.

TIB/RV shows a more consistent performance in all rates, not showing much effect of the changes in the number of publishers. SonicMQ shows a gradual decline in all the rates when there are more messages and publishers.

Test 3 – Connection Time

We are to measure how quickly the subscriber program starts up and receive the first message. There is a constant publisher that is publishing messages before the subscriber is available so that the subscribers do not have to wait for messages to arrive. It measures the time from the program starts up until the time the first message is received.

TIB/RV starts in 0.5 seconds, which is much faster than SonicMQ in 2.55 seconds.

5.2.2. Pub/Sub Test Summary

In the publish/subscribe model, TIB/RV shows a much better performance over SonicMQ. For Test 1, the results show the effect of the performance subject to the changes in the number of publishers and subscribers. TIB/RV was not affected much as a result of the changes, however SonicMQ shows a scalability problem that when more messages are being published and subscribed, the rates are greatly affected.

Finally, TIB/RV takes less effort to start up the subscriber program, resulting in a shorter connection time.

5.2.3. Point-to-Point (Test 2)

Test 2 – Varying the Number of Sender/Receiver Pairs

These tests measure how quickly a MOM can deliver 2000 messages with 10kb sizes, produced in a PTP model. The number of sender/receiver pairs gradually increases to observe the changes. It appears that the sending/receiving rates are much more stable in the TIB/RV than SonicMQ. Again, SonicMQ suffers a scalability problem.

Tests on SonicMQ for sender/receiver pairs with total data size (number of senders x total data by each sender) being too high (greater than 300Mb) crashed with an error with JVM heap size is not big enough. TIB/RV managed to survive the tests without further problems. In order to compare the results, we had to tune down message size and the total messages sent.

Also, more times the tests are repeated, the slower the rates for SonicMQ becomes. This is to do with the lack of RAM in the server. It shows how memory hungry SonicMQ could be.

Figure 16 shows the relationship between the average sending rates to the number of sender/receiver pairs. This is the time it takes to pass the messages from the sender

program to the MOM. It shows gradual decline in the rates for both MOMs. However, SonicMQ appears to decrease faster than TIB/RV.

Figure 17 shows the average receiving rate for both MOMs are being quite constant with the changes in the number of publishers. TIB/RV can receive at an average of approximately 40 MPS, where Sonic receives at 31 MPS. Hence TIB/RV is approximately 33% faster.

Figure 18 shows the total duration of both MOMs for the tests. It appears both MOMs take approximately the same time to send the messages, however TIB/RV takes slightly less time than SonicMQ.

Figure 19 shows the sending rate of both MOMs. It is the time that takes for the messages to travel from the sender program to the receiver program. For 1-2 sender/receiver pairs, SonicMQ is faster than TIB/RV, however, as more sender/receiver pairs are introduced, the rates decreased. This again shows a scalability issue on SonicMQ.

5.2.4. PTP Test Summary

From the test results, TIB/RV still shows better performances. It is much more stable, and not subject to changes in the number of sender/receiver pairs. SonicMQ on the other hand suffers decrease in performance as the increase in the number of sender/receivers. SonicMQ also crashed when the total data sent was too big.

The results from SonicMQ were very inconsistent, sometimes fast, and sometimes slow. We had to repeat the tests a number of times to minimize the error involved. It appears its performance depends greatly on the state of the system (i.e. the amount of free memory available).

Finally, TIB/RV shows outstanding results in handling the overloading of the queues in the PTP Ping Test. Due to the nature of PTP, it is impossible to measure the connection time, as it requires the receiver to be started before a message could be sent.

5.2.5. Test 4 – Memory and CPU Utilization

According to the results obtained, it appears in average that SonicMQ consumes more memory and CPU time than TIB/RV in most tests. This is obvious even without measuring, as throughout the duration of most benchmark tests, the computer becomes significantly slower while SonicMQ is in operation.

This has to do with the native implementation of the MOMs. TIB/RV is implemented in C, and SonicMQ is 100% Java implemented. As discussed earlier, JVM consumes a lot of memory that introduces a lot of overhead to the system. Generally, C is supposed to run a lot faster than Java. More memory and CPU time would be consumed if running multi-cluster configuration in SonicMQ, as it requires another instance of JVM in operation.

As both MOMs vendors did not provide enough information about their underlying implementation, it is difficult to analyse the results for this test further. Perhaps a high-end multiprocessor server with lots of memory will overcome this problem, however it still requires testings before this can be concluded.

5.3. Summary

TIB/RV is the clear winner in the course of the benchmark tests. The results of Tests 1 and 2 show that TIB/RV is faster in both publish/subscribe and point-to-point messaging models. Despite the average publishing rate of SonicMQ is faster, it appears it takes a lot longer to deliver the messages across to the message consumer(s).

SonicMQ consumes more CPU time and memory than TIB/RV. This is because of the difference in their native implementation.

6. Conclusion

The objective of this work was to benchmark two selected MOMs – TIB/RV and SonicMQ by testing their effectiveness in the delivery of messages in publish/subscribe and point-to-point message domains, their program stability and the system resource utilization.

TIB/RV has been in the middleware industry for many years. It is one of the most widely used messaging middleware in the world. It is intended to provide a tool for building and deploying scalable distributed applications faster and easier. The underlying architecture of TIB/RV is the TIB RV Daemon (RVD), which is responsible for the delivery of messages using UDP broadcast.

On the other hand, SonicMQ is a relatively new competitor in the MOM market that is implemented 100% in Java. Being a JMS implementation, it makes it possible for the programs to be portable to other JMS implementations by different vendors without much modification. SonicMQ relies on the broker for message delivery. Applications connect to the SonicMQ broker using TCP, HTTP, and SSL communication protocols.

The results of our benchmark show that TIB/RV has exceptional performance compared to SonicMQ. In summary, they are as follows:

- High publish/subscribe and point-to-point send/receive rates
- High scalability:
  - Publishing rate not affected by introducing more receivers;
  - Subscriber rate increases as more subscribers are introduced.
- Low memory and CPU consumption

The only major downside of TIB/RV is that when there are very few receivers in the network, it could flood the network with many unnecessary UDP packets, introducing congestions.

Benchmark reports provided by Progress quoted SonicMQ's good performance over IBM MQSeries and FioranoMQ on WinNT platform, SonicMQ did not perform as well under the benchmark tests performed in this paper

when using a Linux platform. It was discovered that SonicMQ has the following drawbacks:

- Poor scalability:
  - Performance dampened when more senders and receivers are introduced. This is because the computers are connected in a client-server model, that the server becomes a bottleneck when the system is on stress.
  - Furthermore, the use of TCP connection to connect the applications to the broker enforced a congestion control, limiting the transfer speed.
  - When the total data sent was too high (greater than 300Mb), the system crashed with error: insufficient Java Heap size. However the system is already tuned with largest heap size possible.
- High memory and CPU utilization:
  - SonicMQ is implemented in Java that runs on top of Java Virtual Machine (JVM). JVM was incurs a lot of overheads and resources. As a result, the system's performance was greatly dragged down.

Therefore, TIB/RV is the clear winner over SonicMQ in the benchmark with its outstanding performance, effective speed, high stability and low resource requirement. MOMs with JMS-implementation typically rely on JVM and thus suffer from high overhead, leading to trade off in performance. As JMS is still rather new, it appears there are still quite a few areas that can be improved.

7. References

1. Sun Java Message Service Tutorial, http://java.sun.com/products/jms/tutorial/1_3-fcs/doc/basics.html
2. TIB/Rendezvous Concepts – Product Overview, TIB/Rendezvous Manual
3. TIB/Rendezvous Concepts – Fundamentals, TIB/Rendezvous Manual
4. SonicMQ Deployment Guide, Sonic Software, Page 122, http://www.sonicsoftware.com/products/documentation/deploy.pdf
5. A Benchmark Comparison of Progress SonicMQ and IBM MQSeries, Produced by Point Solutions, http://www.sonicsoftware.com/white_papers/performance.pdf