IBM Message Service Client for .NET

IBM Messaging Service Client (XMS) for .NET Performance Evaluation Version 1.0

IBM Message Service Client for .NET

IBM Messaging Service Client (XMS) for .NET Performance Evaluation Version 1.0

Contents

Tables v

Chapter 1. Preface 1
Target Audience 1
The contents of this SupportPac 1

Chapter 2. Overview 3

Chapter 3. Architecture of Performance Test scenarios 5
Point-to-Point PutGet four queue Scenario 5
Point-to-Point N:N (multiple producers, consumers, queues) scenario 6
Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) scenario. 6
Publish Subscribe N:N (multiple publishers, subscribers, topics) scenario 7

Chapter 4. Test Scenarios 9
Point-to-Point four queue PutGet scenario 9
Non-Persistent messages 9
Persistent messages 9
Point to Point N:N (multiple producers, consumers, queues) scenario 11
Non Persistent results 11
Persistent Results 13
Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) scenario 17
Binding mode results 17
Managed Client mode results 18
Unmanaged client mode results 20
Publish Subscribe N:N (multiple publishers, subscribers, topics) scenario 21
Non-persistent results 22
Persistent results 24

Chapter 5. Testing with WebSphere MQ V7 client features 27
Performance comparison by varying ShareCnv . . 27
Performance comparison by using Asynchronous Put 28
Performance comparison using Read Ahead . . . 28
Performance comparison by varying Read Ahead buffer Size. 30

Chapter 6. Tuning and Programming guidelines 31
Selecting the Log location 31
Level of log write 31
Type of logging 31
Log file extent size 31
Number of log file extents 32
LogBufferPages 32
Setting the Log parameter values in the registry . . 32
Queue buffer sizes 32
Queue manager channels 33
Enabling Prefetch in the system 33
Enabling Asynchronous Put 33
Enabling Read Ahead 34
Setting the ReadAhead buffer size 34
Disabling multiplexing and shared conversation . . 34
Send and Receive Buffer size for large messages . . 35

Chapter 7. Machine and Test Configuration 37

Chapter 8. Appendix 39

Notices 41

Trademarks and service marks 43

Index 45

Tables

1. Test Parameters 3
2. Point-to-Point 4Q Binding mode maximum rate 10
3. Point-to-Point 4Q Managed Client mode maximum rate 10
4. Point-to-Point 4Q Unmanaged client mode maximum rate 10
5. Point-to-Point N:N Non Persistent maximum rate 13
6. Point-to-Point N:N Persistent maximum rate 16
7. Publish Subscribe 1:N Binding mode maximum rate 18
8. Publish Subscribe 1:N Managed Client mode maximum rate 20
9. Publish Subscribe 1:N Unmanaged client mode maximum rate 21
10. Publish Subscribe N:N Non Persistent mode maximum rate 23
11. Publish Subscribe N:N Persistent mode maximum rate 26
12. Performance Comparison by varying ShareCnv 27
13. Performance comparison by using Asynchronous Put 28
14. Performance maximum enabling Read Ahead 29
15. Performance comparison by varying Read Ahead buffer size 30

Chapter 1. Preface

This report presents a performance evaluation of XMS .NET on a Windows platform with WebSphere® MQ. The report is intended to assist in capacity planning, and application design.

Target Audience

This SupportPac is designed to enable users to:

- Design an XMS .NET solution with WebSphere MQ.
- Understand the scalability of XMS .NET applications.
- Leverage the advantages of XMS .NET features for better performance.

The contents of this SupportPac

The contents of this SupportPac cover:

- Charts and Tables providing information about XMS.NET performance scalability.
- Performance comparison using WebSphere MQ V7 features.
- Advice on programming and tuning.

Chapter 2. Overview

The purpose of this report is to provide a reference on XMS .NET performance for various scenarios. Commonly used customer scenarios, including point-to-point, and publish-subscribe were measured. The following scenarios were tested:

1. Point to Point four queue PutGet.
2. Point to Point N:N multiple producers, consumers.
3. Publish Subscribe 1:N single publisher, multiple subscribers.
4. Publish Subscribe N:N multiple publishers, subscribers.

The XMS .NET Performance Harness tool was used to conduct the tests. See XMS .NET Performance Harness documentation (available at https://www.ibm.com/developerworks/mydeveloperworks/files/app/collection/5bd0fa23-4704-44dc-a5d5-ffe7cd205bf3?lang=en) for information about the XMS .NET Performance Harness tool.

The XMS .NET Performance Harness tool performs the functions of PutGet, Sender, Receiver, Publisher, Subscriber applications as needed. These tests were run in a .NET environment.

The performance test scenarios are adopted from JMS Performance Evaluation document MP07. The XMS .NET Performance Harness applications and Queue Manager are hosted on a single system.

In the context of each test, following is applicable:

Table 1. Test Parameters

Message size	2048 bytes
Message type	Text message
Delivery mode used	Persistent and Non-persistent messages
Mode of test	Binding, Managed client (.NET managed client), Unmanaged client (.NET unmanaged client)
Input Rate for Scenario 2 and 4	Constant input rate for each application thread
Input Rate scenario 1 and 3	Maximum input rate
Test duration	300 seconds
Discard period	The first 10 seconds of data are discarded
ReadAhead	Enabled for receiver and subscriber applications
Sharing of TCP/IP channel instance for conversation	The number of connections to share is set 1.
AsynchronousPut	Enabled for the sender and publisher applications
Transaction property for sessions.	Non transactional for all tests.
Log and Queue Buffer tuning	Done
Windows OS tuning	Prefetch feature is enabled and set to 3

The test results are presented in charts and tables.

To understand the benefits of WebSphere MQ V7 features, tests were run with selected features of WebSphere MQ enabled, and then disabled. The results are presented in charts and tables.

Chapter 3. Architecture of Performance Test scenarios

Four types of application scenarios were created to conduct performance testing. Two of these scenarios use Point-to-Point, and two scenarios use Publish-Subscribe.

- The Point to Point four queue, (four queues) PutGet scenario: In this scenario, applications are allocated to one of the four queues. The performance impact of several applications interacting with a limited number of queues is measured.
- The Point to Point N:N (multiple producers, consumers, queues) scenario: In this scenario, the number of triplets (producer, consumer, and queue) are sequentially increased. This test measures the scalability of the system when the triplets (producer, consumer, and queue) are increased.
- The Publish-Subscribe 1:N (single publisher, multiple subscribers) scenario: This scenario measures the system capacity when the number of subscribers is increased.
- The Publish Subscribe N:N (multiple publishers, subscribers, topics) triplets scenario: This test measures the system capacity when the number of triplets (publisher, subscriber, topic) is sequentially increased.

Point-to-Point PutGet four queue Scenario

In this scenario, a PutGet application is used to measure performance. In the PutGet application, each thread is allocated to one of the four queues. The thread that puts the message to a particular queue gets the message back from the same queue.

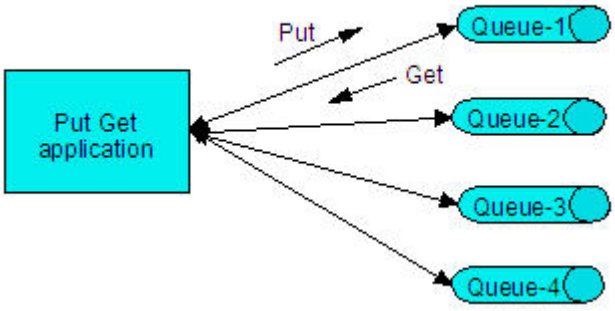

Figure 1. Point to Point, 4Q Put/Get scenario

- The number of applications is increased by increasing the number of application threads.
- During the test, each thread measures the number of PutGet sequences completed for each test interval. The total number of completed PutGet sequences for all the threads shows the impact on performance versus the number of active application threads.

Point-to-Point N:N (multiple producers, consumers, queues) scenario

In this scenario, two applications are used. One application is the producer, and the other is the consumer. Multiple threads are used in the producing and consuming applications. Each pair of application threads communicate through a specific single queue.

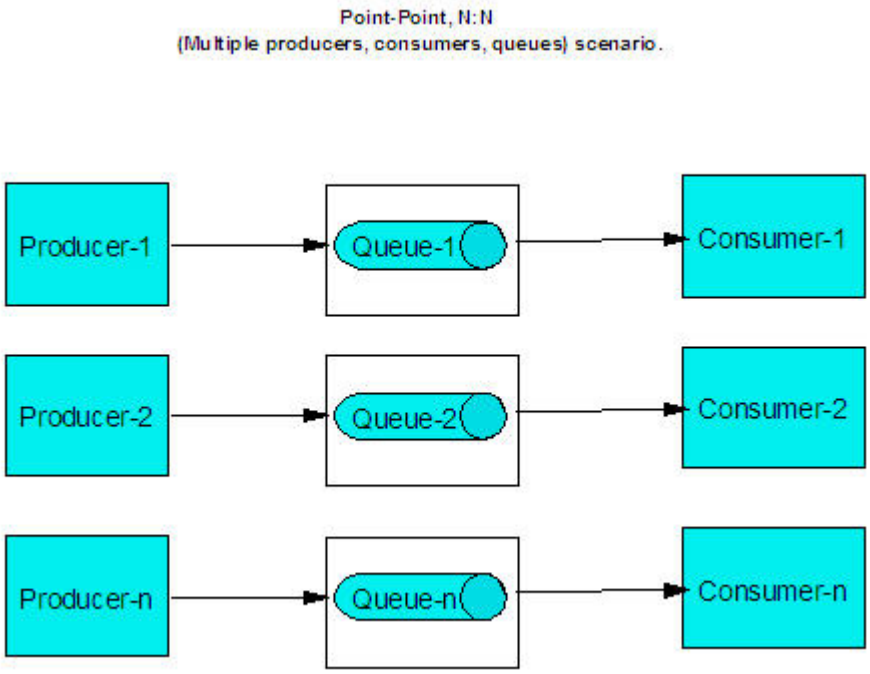

Figure 2. Point to Point, N:N (Multiple producers, consumers, queues) scenario

- Both the applications are on the same system. The message producer sends messages at constant rate. The number of triplets (producer, consumer, and queue) is gradually increased. The performance impact is measured as the number of triplets is increased. The operation of the producing application putting the messages, and the receiving application retrieving the message is counted as two messages.
- The sum of the sender rate and receiver rate for all the applications gives the performance capability of this configuration.
- Non persistent messages: 3200 messages/sec/application thread
- Persistent messages: 400 messages/sec/application thread
- The message size is 2048 bytes.

Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) scenario

In this scenario, the Publisher publishes messages on a single topic. The number of subscribers to the topic is gradually increased from 2 to 40.

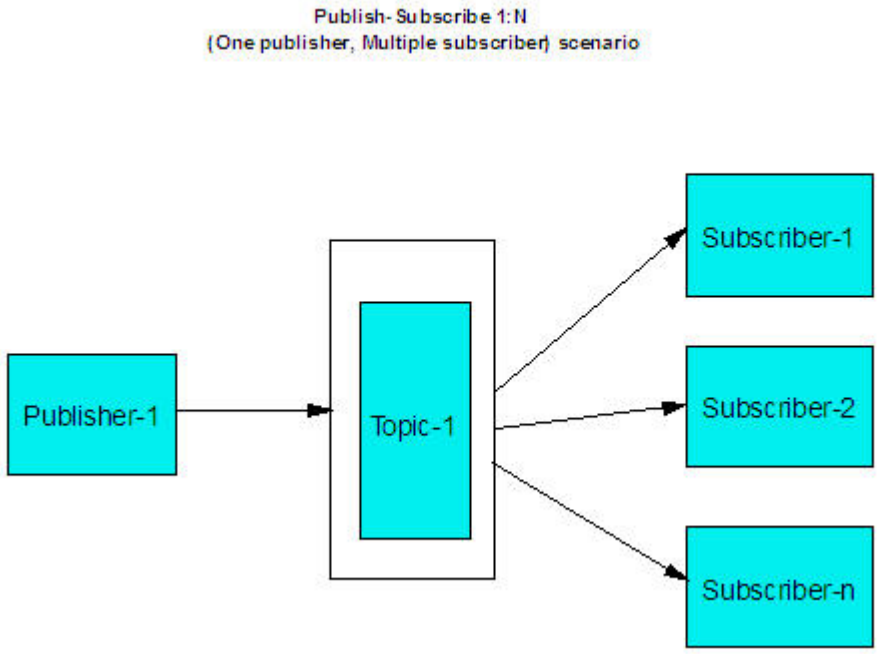

Figure 3. Publish Subscribe 1:N (Single publisher, Single topic, multiple subscribers) scenario

- The publisher produces messages as fast as possible. The number of subscribers is increased. The impact on performance is measured.
- The sum of messages published, and sum of messages received by the subscribers, gives the performance statistics for this configuration.

Publish Subscribe N:N (multiple publishers, subscribers, topics) scenario

In this scenario, a Publisher publishes messages for a particular topic. A single Subscriber subscribes to that topic to get the messages.

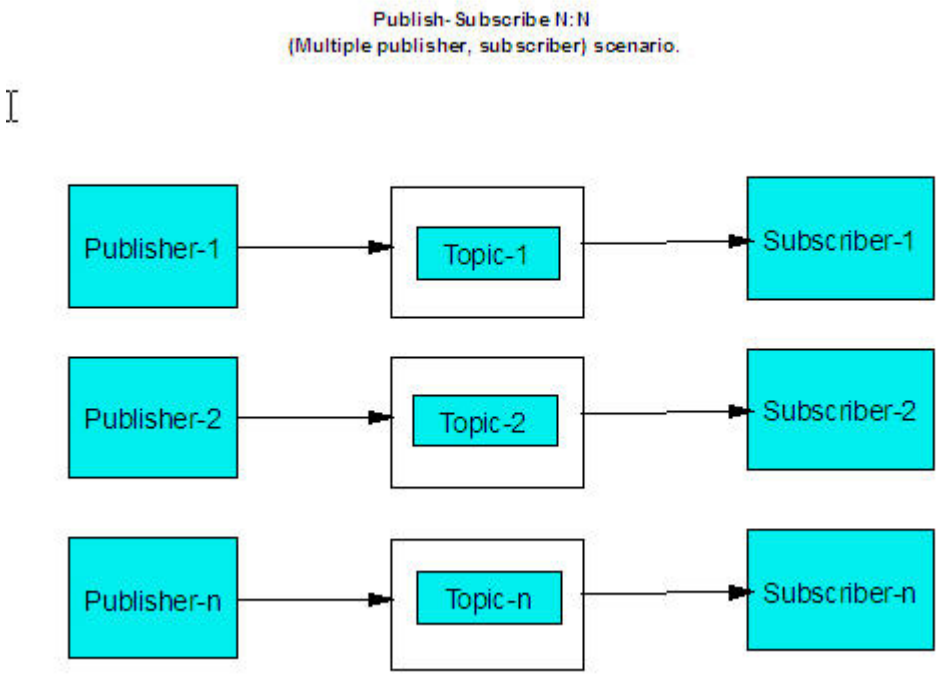

Figure 4. Publish Subscribe N:N (multiple publishers, subscribers, topics) scenario

- The triplet (publisher, subscriber, topic) is gradually increased up to 10.
- The sum of all messages published, and sum of all messages received by the subscribers gives the total performance capacity of this configuration.
- The Publisher publishes messages at a constant rate.
- Non persistent messages: 1600 messages/sec/application thread.
- Persistent messages: 800 messages/sec/application thread.
- The message size is 2048 bytes.

Chapter 4. Test Scenarios

The performance results of tests conducted for four scenarios are given below. Test results are explained using charts and tables.

Point-to-Point four queue PutGet scenario

In the Point-to-Point 4Q Put/Get scenario, the number of application threads that Put and Get messages, is gradually increased from 2 to 40. Message throughput, and CPU utilization are measured in each interval. The complete operation of putting the message, and getting back the message is considered as one message.

Non-Persistent messages

The graph in figure 5 displays the results of Point-to-Point four queue PutGet Non-Persistent test, conducted in binding mode, managed client mode, and unmanaged client mode.

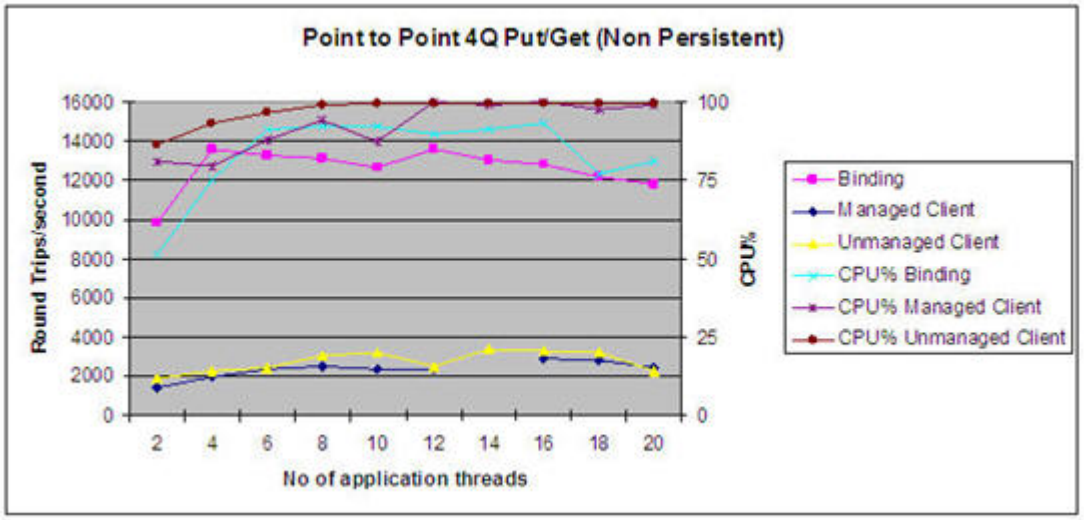

Figure 5. 5 Point-to-Point 4Q Non Persistent chart

During the test, the performance, measured as round trips per second, gradually increases with number of applications until CPU utilization reaches maximum. Then the performance remains fairly constant as the number of application thread increases.

Persistent messages

The graph in Figure 6 displays the results of Point-to-Point PutGet four queue persistent test conducted in binding mode, managed client mode, and unmanaged client mode.

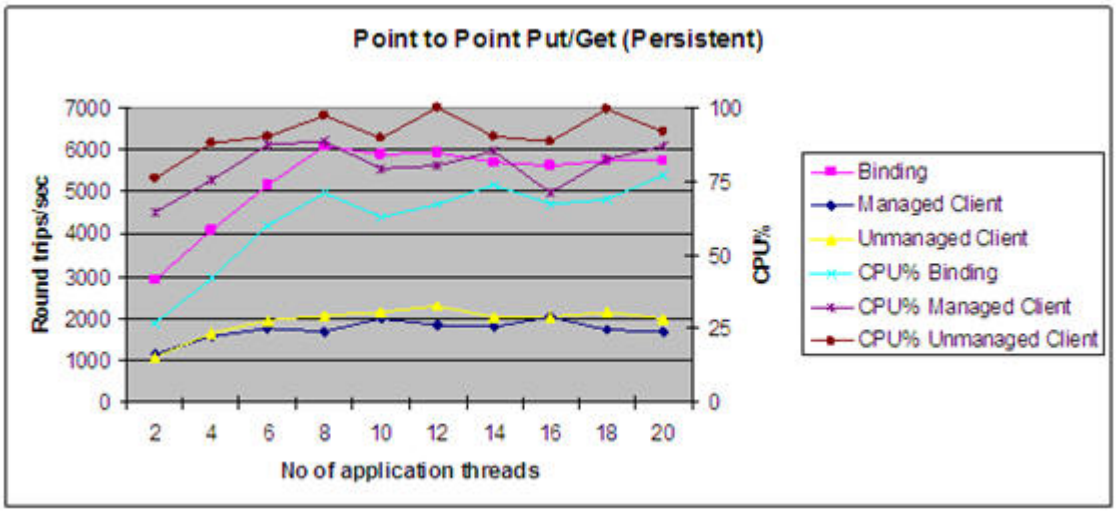

Figure 6. Point-to-Point 4Q Persistent chart

During the test, the performance gradually increases with number of applications until CPU utilization reaches maximum. Then the performance remains fairly constant with increase in application threads.

The maximum performance rate achieved in binding mode is shown in Table 2. The corresponding CPU utilization is shown.

Table 2. Point-to-Point 4Q Binding mode maximum rate

Binding mode	No of application threads	Max-rate in roundtrips/sec	CPU%
Non persistent	12	13544	90
Persistent	8	6086	71

The maximum performance rate achieved in managed client mode is shown in Table 3. The corresponding CPU utilization is provided.

Table 3. Point-to-Point 4Q Managed Client mode maximum rate

Managed Client mode	No of application threads	Max-rate in roundtrips/sec	CPU%
Non persistent	16	2894	100
Persistent	16	2018	71

The maximum performance rate achieved in unmanaged client mode is shown in Table 4. The corresponding CPU utilization is provided.

Table 4. Point-to-Point 4Q Unmanaged client mode maximum rate

Unmanaged Client mode	No of applications	Max-rate in roundtrips/sec	CPU%
Non persistent	12	3343	99.6
Persistent	26	2282	100

Point to Point N:N (multiple producers, consumers, queues) scenario

In these tests, the producing application generates messages at constant rate. The number of triplets (producer, consumer, and queue) in the configuration is increased from 1 to 10.

The performance results, and CPU utilization, is captured for each of the tests.

Non Persistent results

The non-persistent tests are conducted with a constant input message rate of 3200 messages per second for each producer application.

Binding mode

The graph in Figure 7 displays the results of Point-to-Point N:N (multiple producers, consumers, queues) Non-Persistent test, conducted in binding mode.

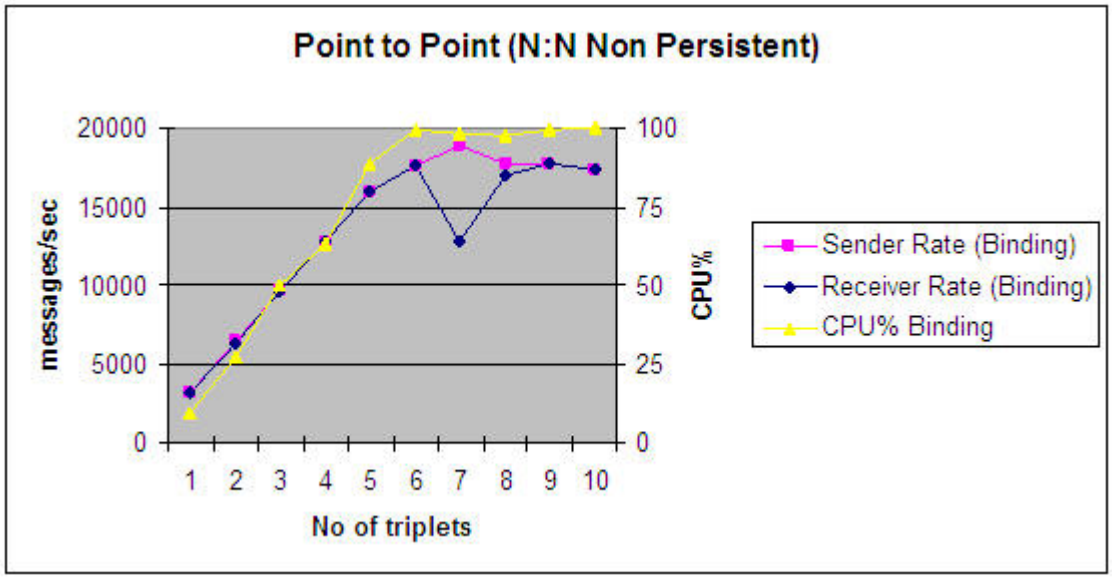

Figure 7. Point-to-Point N:N Non Persistent Binding chart

During the test, the receiver rate increases linearly with the sender rate until the CPU utilization reaches maximum. Then, the sender rate and receiver rate remain almost constant.

Managed Client Mode

The graph in Figure 8 displays the results of Point-to-Point N:N (multiple producers, consumers, queues) Non-Persistent test, conducted in managed client mode.

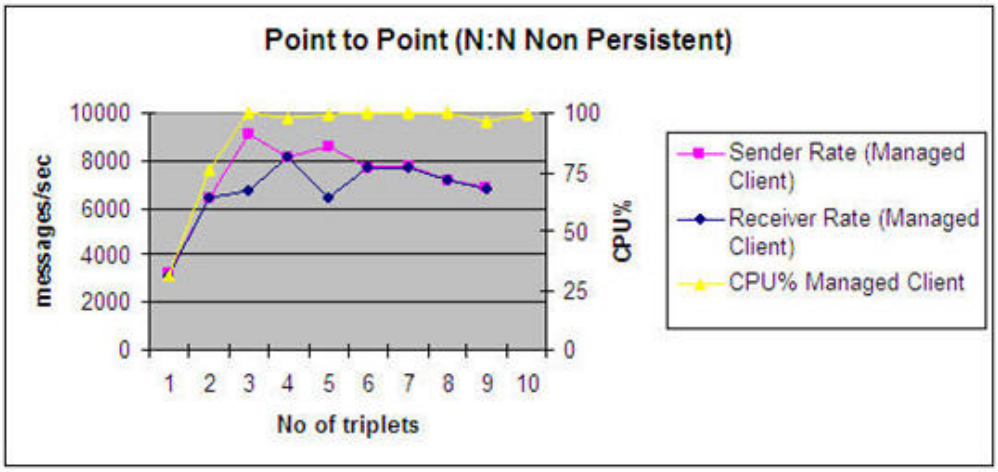

Figure 8. Point-to-Point N:N Non Persistent, Managed client chart

During the test, the receiver rate increases linearly with the sender rate until the CPU utilization reaches maximum. Then, the sender rate and receiver rate remains almost constant.

Unmanaged Client mode

The graph in Figure 9 displays the results of Point-to-Point N:N (multiple producers, consumers, queues) Non-Persistent test, conducted in unmanaged client mode.

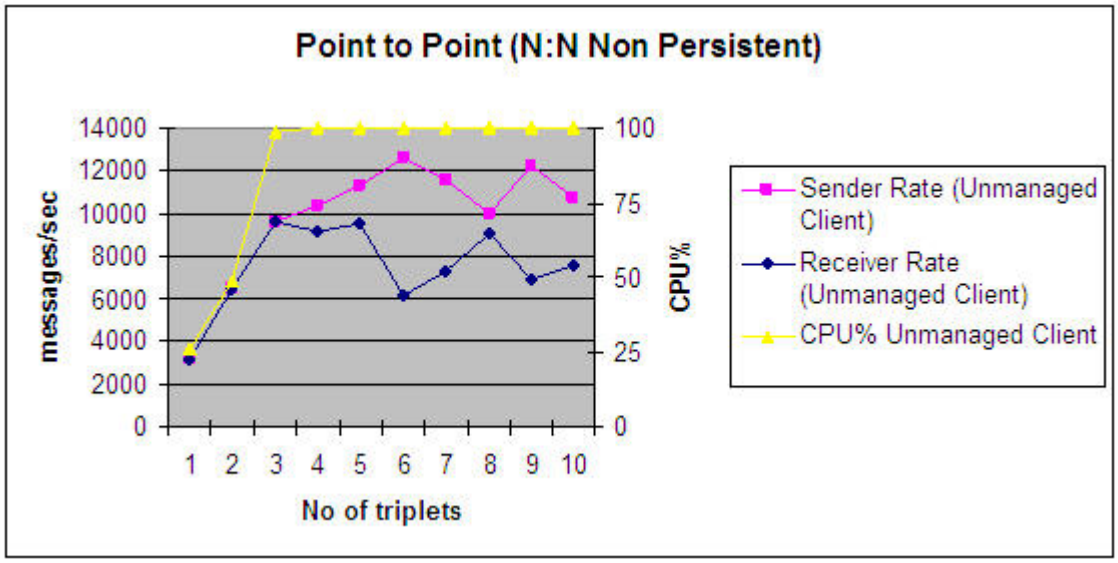

Figure 9. Point-to-Point N:N Non Persistent Unmanaged client chart

- During the test, the receiver rate increases linearly with the sender rate until the CPU utilization reaches maximum. Further increases to the sender rate results in a decrease to the receiver rate.
- When the receiver is not able to keep up with the messages being sent by the producer, the messages queue.

The best performance statistics achieved using this configuration for non-persistent messages in binding, client, and unmanaged client modes is given in Table 5.

Table 5. Point-to-Point N:N Non Persistent maximum rate

Non persistent	Binding	Managed Client	Unmanaged Client
Number of triplets	6	4	3
Receiver rate in messages/sec	17545	8112	9616
Sender rate in messages/sec	17567	8159	9600
Total rate in messages/sec	35112	16271	19216
Expected rate in messages/sec	38400	25600	19200
CPU%	99.2	98	98.43

The total rate is the sum of the sender rate and receiver rate.

The expected rate is calculated as:

Expected rate = 3200 * number of triplets * 2.

Persistent Results

The persistent tests are conducted with constant input message rate of 400 messages per second for each producer application.

Binding mode

The graph displays the results of Point-to-Point N:N (multiple producer, consumer, queue) Persistent test, conducted in binding mode.

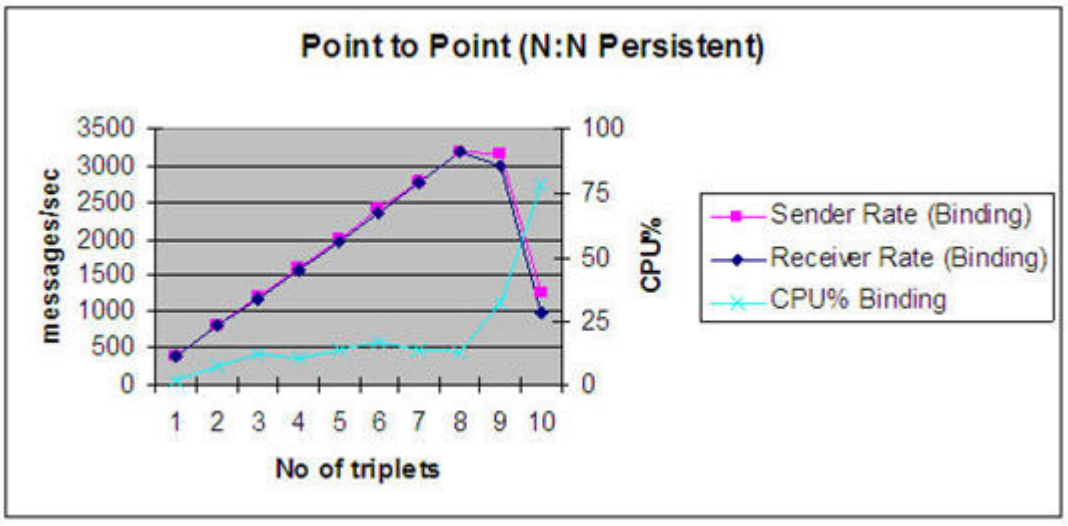

Figure 10. Point-to-Point N:N Persistent, Binding chart

During the test, the sender and receiver rate increase linearly up to eight triplets. After that, there is a sudden drop in performance, and also in CPU utilization. This drop is attributed to the increase in the number of input/output activities waiting to be completed.

The Average Disk Queue Length variation chart, corresponding to Point-to-Point N:N Persistent test in binding mode, is provided in Figure 11. The Average Disk Queue Length indicates the number of input/output activities waiting to be completed.

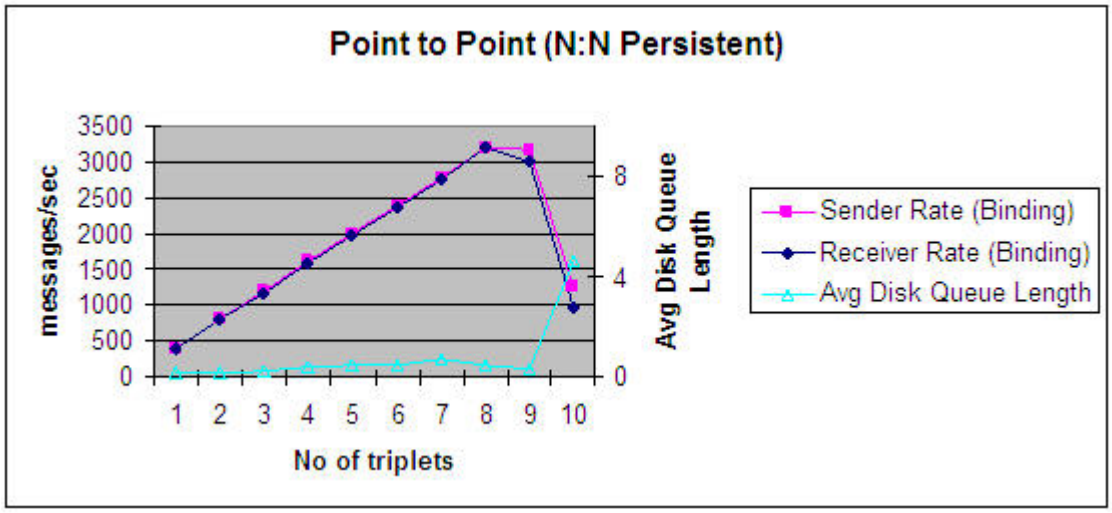

Figure 11. Point-to-Point N:N Persistent, Binding, Average Disk Queue Length variation

Managed Client mode

The graph in Figure 12 displays the results of Point-to-Point N:N (multiple producers, consumers, queues) Persistent test, conducted in managed client mode.

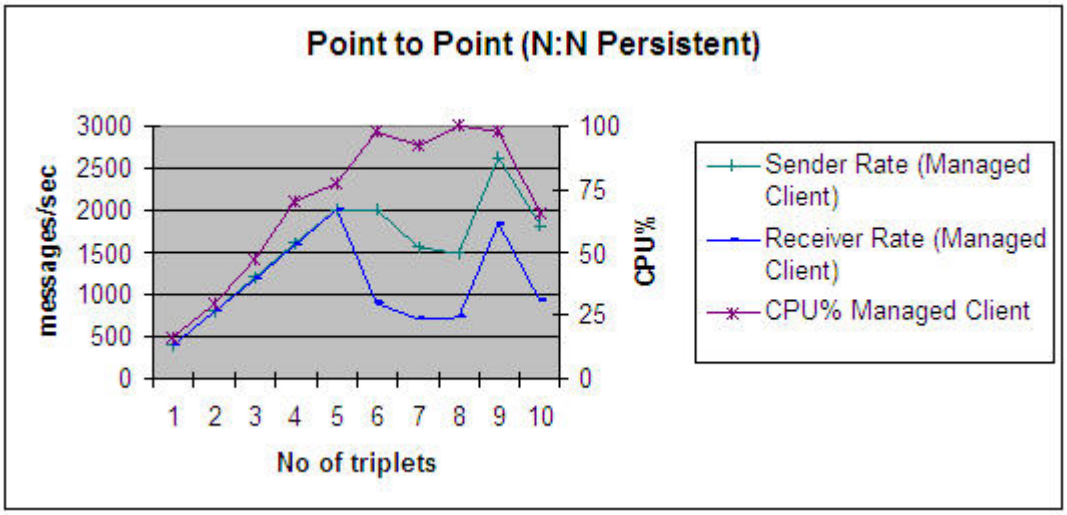

Figure 12. Point-to-Point N:N Persistent, Managed client chart

The sender and receiver rate increase linearly up to five triplets. After that, there is a drop in performance, and also in CPU utilization. This drop is attributed to the increase in the number of input/output activities waiting to be completed.

The Average Disk Queue Length variation chart, corresponding to Point-to-Point N:N Persistent test in managed client mode, is provided in Figure 13. The Average Disk Queue Length gives an indication of the number of input/output activities waiting to be completed.

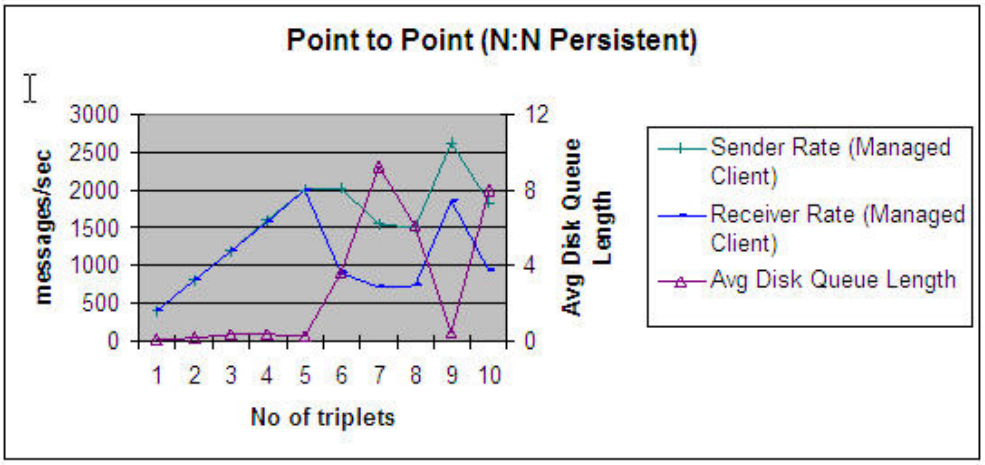

Figure 13. Point to Point N:N Persistent, Managed Client, Average Disk Queue Length variation

Unmanaged Client mode

The graph in Figure 14 displays the results of Point-to-Point N:N (multiple producers, consumers, queues) Persistent test, conducted in unmanaged client mode.

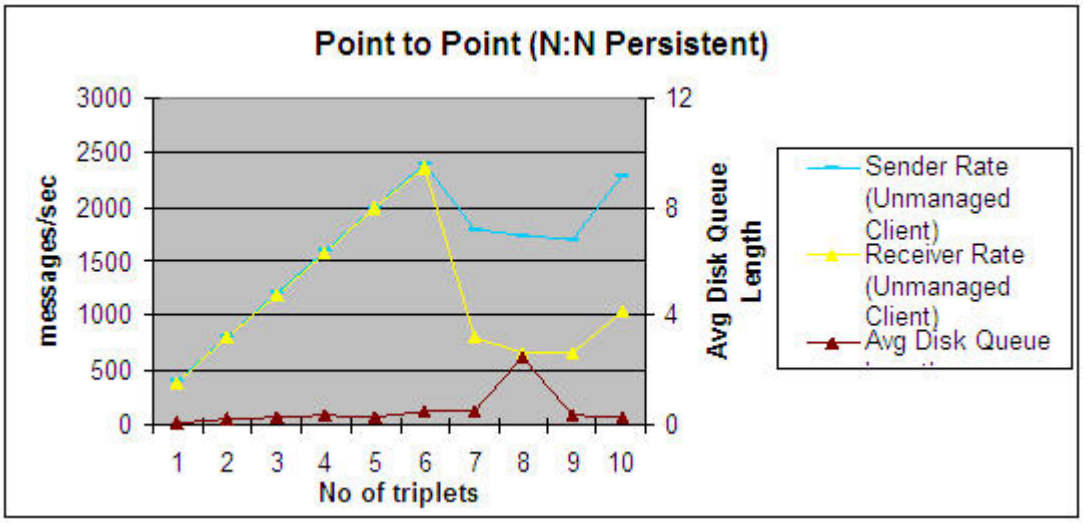

Figure 14. Point-to-Point N:N Persistent, Unmanaged client chart

- During the test, the sender and receiver rate increase linearly up to six triplets. After that, there is a drop in performance, and also in CPU utilization. This drop is attributed to the increase in the number of input/output activities waiting to be completed.

- The Average Disk Queue Length variation corresponding to Point-to-Point N:N Persistent, unmanaged client chart is provided. The Average Disk Queue Length gives an indication of the number of input/output activities waiting to be completed.
- When the receiver is not able to keep up with the messages being sent by the producer, the messages queue. Queuing indicates that superior resources are needed to increase the throughput.

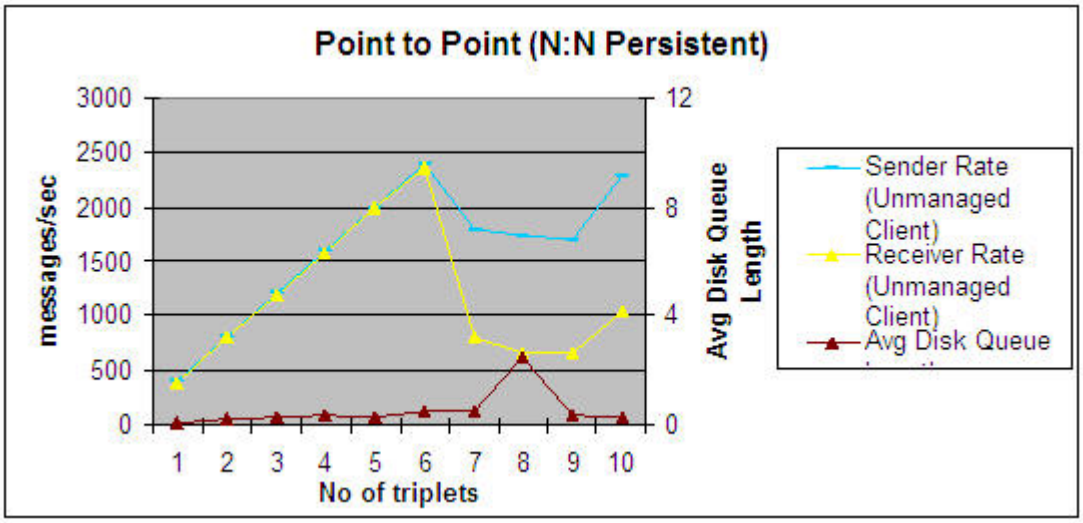

Figure 15. Point to Point N:N Persistent, unmanaged client, Average Disk Queue Length variation

The maximum performance achieved using this configuration for persistent messages in binding, client, and unmanaged client modes is given in Table 6.

Table 6. Point-to-Point N:N Persistent maximum rate

Persistent	Binding	Managed Client	Unmanaged Client
Number of triplets	8	5	6
Receiver rate in messages/sec	3192	1972	2359
Sender rate in messages/sec	3197	1999	2400
Actual rate in messages/sec	6389	3971	4759
Expected rate in messages/sec	6400	4000	4800
CPU%	12.5	79.6	83.2

The total rate is the sum of sender rate and receiver rate.

The expected rate is calculated as:

Expected rate = 400 * number of triplets * 2.

Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) scenario

In these tests, the publisher produces messages at a maximum rate. Initially, one publisher and two subscribers are used. Then the number of subscribers is then increased to 40.

The performance results, and CPU utilization, are captured for each of the tests.

Binding mode results

The results of Publish Subscribe 1:N (Single publisher, Single topic, multiple subscribers) Non Persistent test, conducted in Binding mode.

Non Persistent messages

The graph in Figure 16 displays the results of the Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) Non Persistent test, conducted in Binding mode.

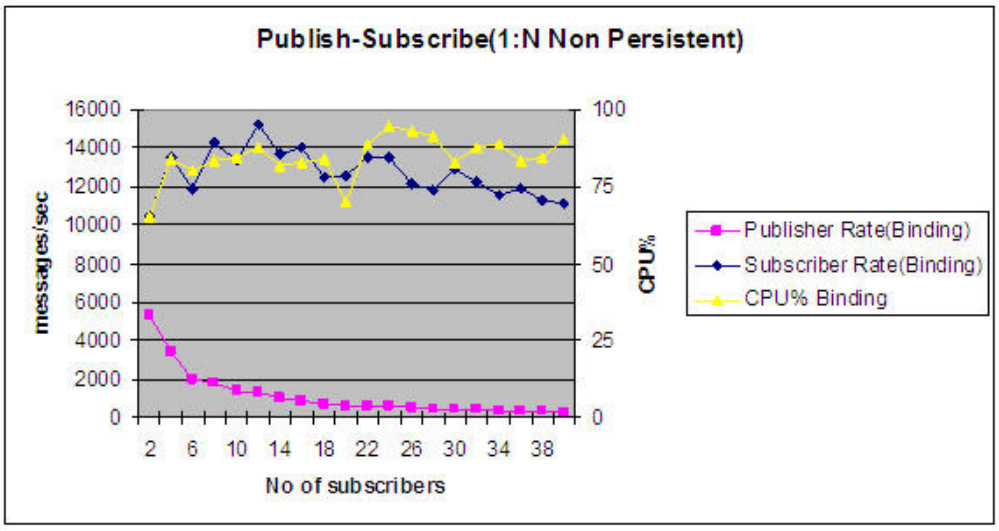

Figure 16. Publish Subscribe 1:N Non Persistent, Binding chart

During the test, as the number of subscribers increase, the publisher gets less CPU time for publishing messages. Hence, the rate of publishing drops with the increase in the number of subscribers.

Persistent messages

The graph in Figure 17 displays the results of Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) Persistent test, conducted in Binding mode.

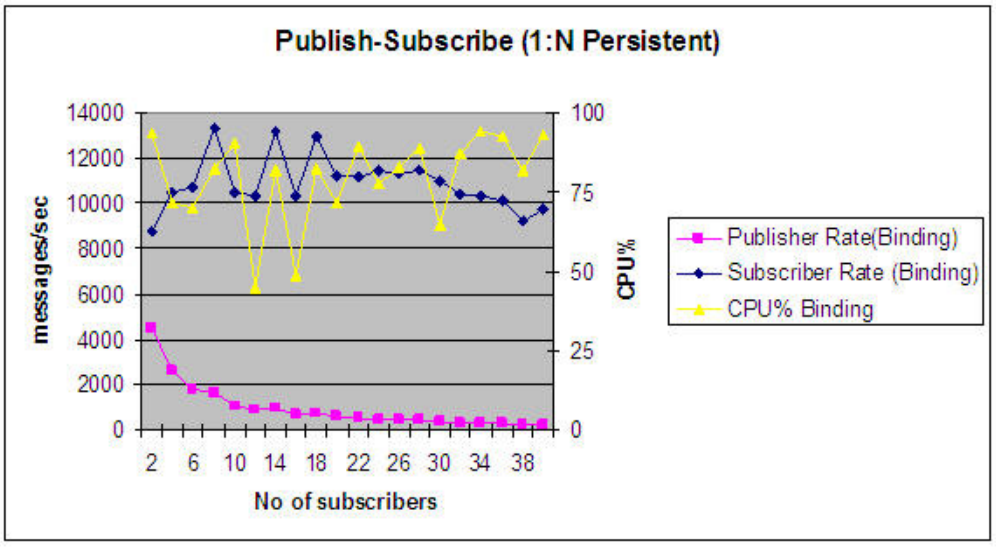

Figure 17. Publish Subscribe 1:N Persistent, Binding chart

During the test, as the number of subscribers increase, the publisher gets less CPU time for publishing the messages. Hence, the rate of publishing drops with increase in number of subscribers.

The maximum rate achieved for non-persistent and persistent messages in binding mode is given in Table 7.

Table 7. Publish Subscribe 1:N Binding mode maximum rate

Binding mode	No of subscribers	Publish rate in messages/ sec	Subscribe rate in messages/ sec	Expected subscriber rate in messages/ sec	Actual total rate in messages/ sec	CPU%
Non persistent	4	3396	13557	4 * 3396 = 13584	16593	83.9
Persistent	8	1669	13305	8 * 1669 = 13352	14974	82.4

The total rate is the sum of the publisher and subscriber rate.

The expected subscriber rate is calculated as:

Expected subscriber rate = publish rate * number of subscribers.

Managed Client mode results

Non Persistent messages

The graph in Figure 18, displays the results of Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) Non Persistent test, conducted in Managed Client mode.

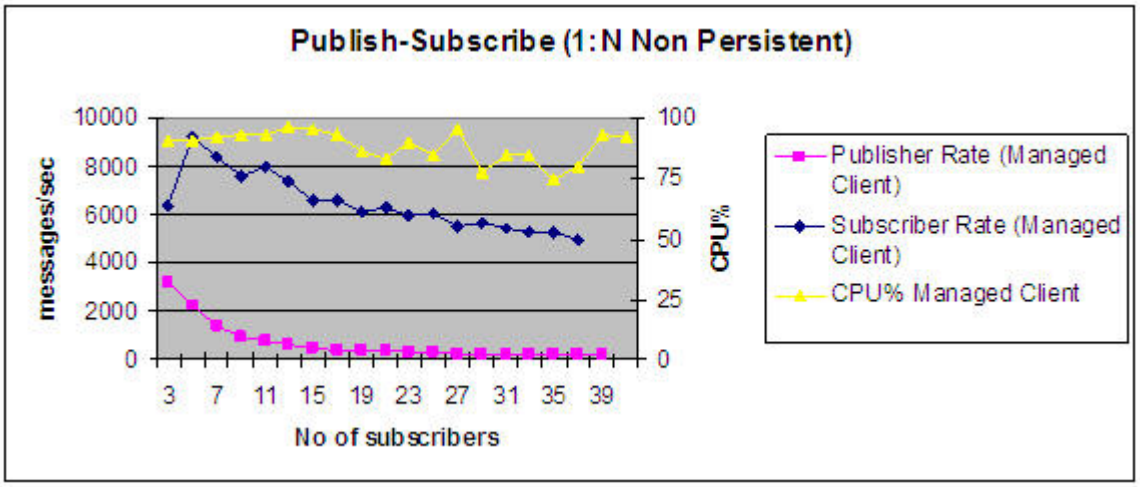

Figure 18. Publish Subscribe 1:N Non Persistent, Managed client chart

During the test, as the number of subscribers increase, the publisher gets less CPU time for publishing the messages. Hence, the rate of publishing drops as the number of subscribers increase.

Persistent results

The graph in Figure 19, displays the results of Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) Persistent test, conducted in Managed Client mode

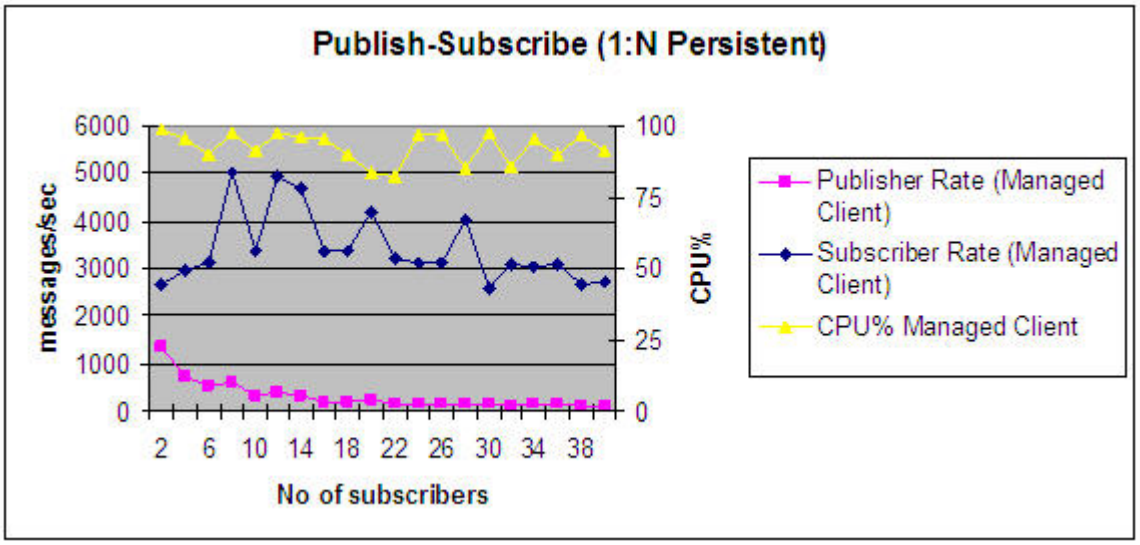

Figure 19. Publish Subscribe 1:N Persistent, Managed client chart

During the test, as the number of subscribers increase the publisher gets less CPU time for publishing the messages. Hence, the rate of publishing drops with an increase in subscribers.

The maximum rate achieved for non-persistent and persistent messages in Managed client mode is given in Table 8.

Table 8. Publish Subscribe 1:N Managed Client mode maximum rate

Managed Client mode	No of subscribers	Publish rate in messages/ sec	Subscribe rate in messages/ sec	Expected subscribe rate	Total rate in messages/ sec	CPU%
Non persistent	4	2269	9242	4 * 2269 = 9076	11511	90.7
Persistent	8	618	4999	8 * 618 = 4944	5617	97.6

The total rate is the sum of the publisher and subscriber rate.

The expected subscriber rate is calculated as:

Expected subscriber rate = publish rate * number of subscribers.

Unmanaged client mode results

Non-Persistent messages

The graph in Figure 20, displays the results of Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) Non-Persistent test, conducted in unmanaged client mode.

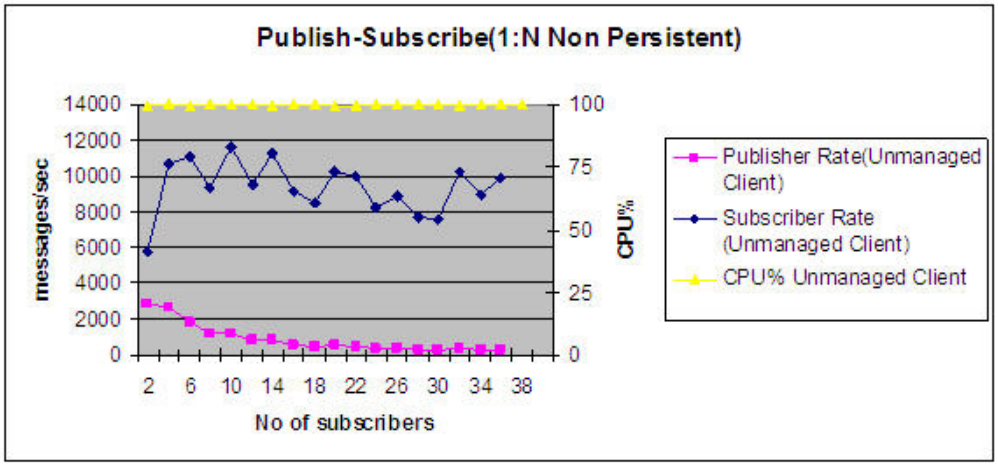

Figure 20. Publish Subscribe 1:N Non Persistent, Unmanaged client chart

As the number of subscribers increases, the publisher gets less CPU time for publishing the messages. Hence, the rate of publishing drops with an increase in the number of subscribers.

Persistent messages

The graph in Figure 21, displays the results of Publish Subscribe 1:N (single publisher, single topic, multiple subscribers) Persistent test, conducted in unmanaged client mode.

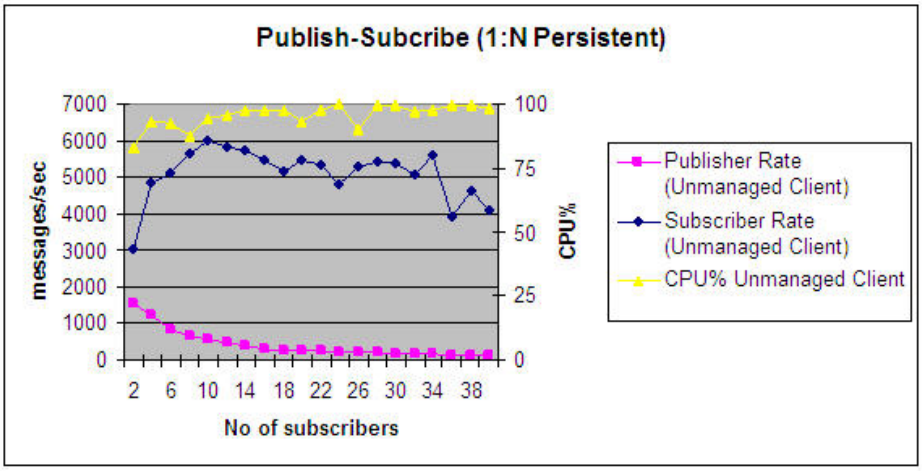

Figure 21. Publish-Subscribe (1: N Persistent), Unmanaged client chart

During the test, as the number of subscribers increase, the publisher gets less CPU time for publishing the messages. Hence, the rate of publishing drops with an increase in the number of subscribers.

The maximum rate achieved for non-persistent and persistent messages in unmanaged client mode is given in Table 9.

Table 9. Publish Subscribe 1:N Unmanaged client mode maximum rate

Unmanaged client mode	No of subscribers	Publish rate in messages/ sec	Subscribe rate in messages/ sec	Expected subscribe rate in messages/ sec	Total rate in messages/ sec	CPU%
Non persistent	4	2675	10731	4* 2675 = 10700	13406	100
Persistent	10	582	6003	10 * 582 = 5820	6585	94.6

The total rate is the sum of publisher rate and subscriber rate.

The expected subscriber rate is calculated as:

Expected subscriber rate = publish rate * number of subscribers.

Publish Subscribe N:N (multiple publishers, subscribers, topics) scenario

In these tests, the publisher publishes at constant rate. The number of triplets (publisher, subscriber, topic) is gradually increased to 10. The performance, measured in messages per second, and CPU utilization, is captured for each of the tests.

Non-persistent results

For non-persistent tests, each publisher publishes 1600 publications/sec.

Binding mode

The graph displays the results of Publish Subscribe N:N (multiple publishers, topics, subscribers) Non-Persistent test, conducted in binding mode.

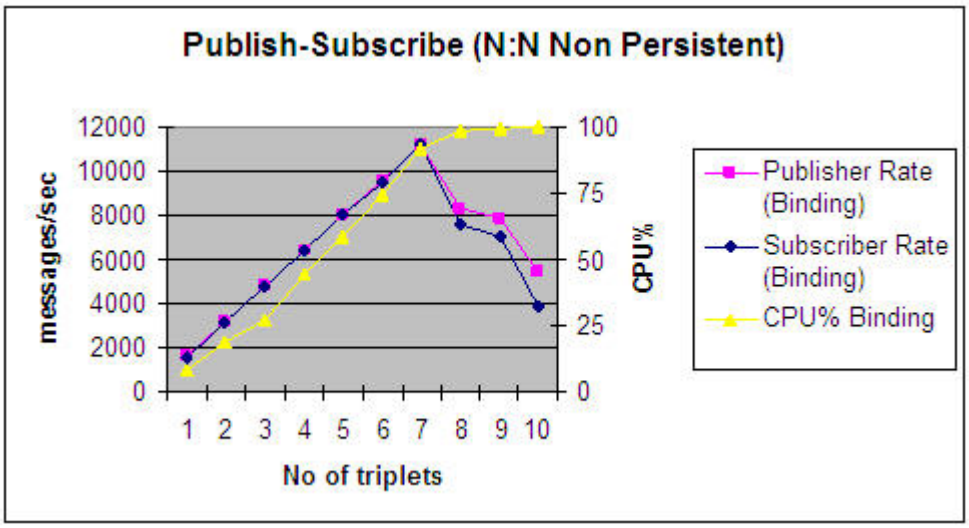

Figure 22. Publish Subscribe N:N Non Persistent, Binding chart

The publisher rate and subscriber rate linearly increase until the CPU utilization reaches maximum. After that, the performance reduces with an increase in the number of applications.

Managed Client mode

The graph displays the results of Publish Subscribe N:N (multiple publishers, topics, subscribers) Non-Persistent test, conducted in managed client mode.

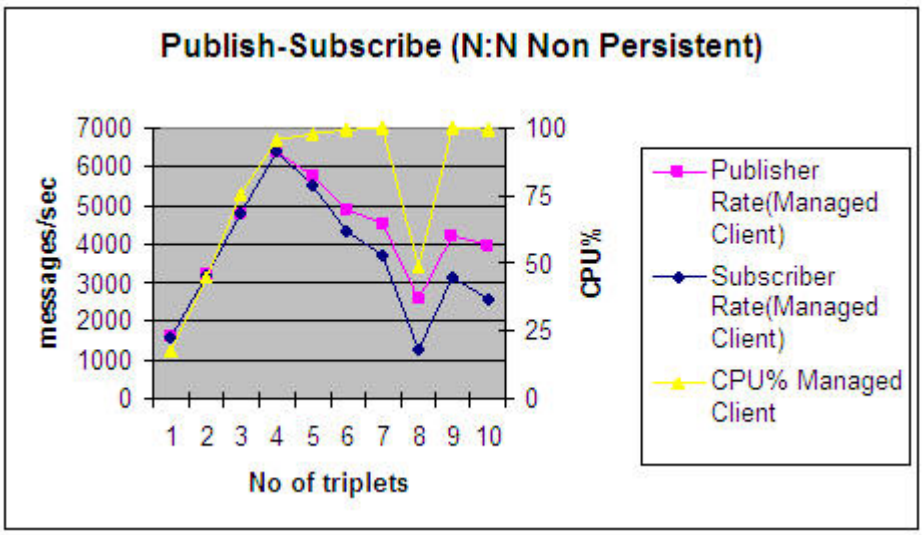

Figure 23. Publish Subscribe N:N Non Persistent, Managed Client chart

During the test, the publisher rate and subscriber rate linearly increase until the CPU utilization reaches maximum. After that, the performance reduces with the increase in the number of applications.

Unmanaged client mode

The graph displays the results of Publish Subscribe N:N (multiple publishers, topics, subscribers) Non-Persistent test, conducted in unmanaged client mode.

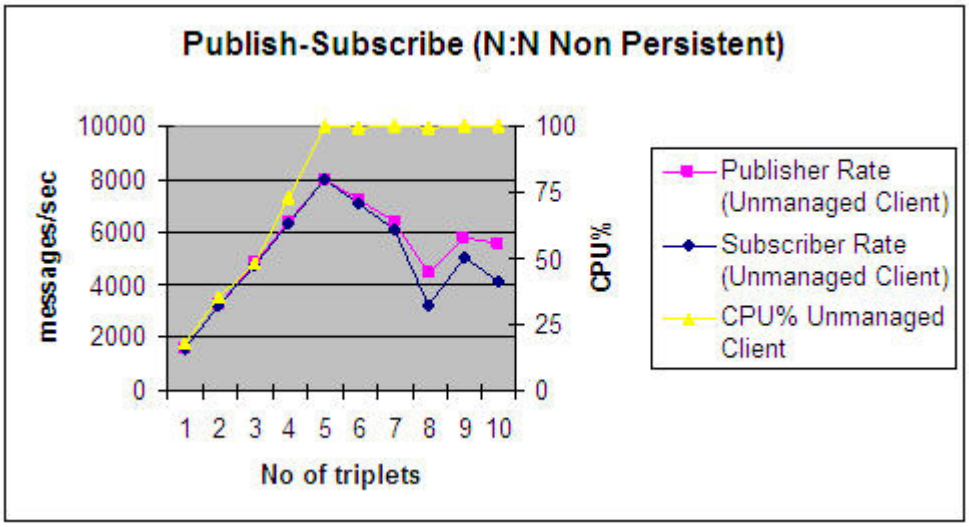

Figure 24. Publish Subscribe N:N Non Persistent, Unmanaged client chart

During the test, the publish rate and subscriber rate linearly increase until the CPU utilization reaches maximum. After that, the performance reduces with an increase in the number of applications.

When the subscriber is not able to keep up with the messages being published by the publisher, the messages queue.

The maximum performance that can be achieved with the existing resource for non-persistent messages in binding, client, and unmanaged client modes is given in the table.

Table 10. Publish Subscribe N:N Non Persistent mode maximum rate

Non persistent	No of triplets	Publish rate in messages/ sec	Subscribe rate in messages/ sec	Expected total rate in messages/ sec	Actual total rate in messages/ sec	CPU%
Binding	7	11199	11212	22400	22414	91.7
Client	4	6399	6402	12800	12801	95.3
Unmanaged	5	8000	8000	16000	15986	100

The total rate is the sum of publisher rate and subscriber rate.

The total expected rate is calculated as

Total expected rate = 1600 * number of triplets * 2.

Persistent results

For persistent tests, the publisher publishes at a rate of 800 publications per second for each application.

Binding mode

The graph in Figure 25, displays the results of Publish Subscribe N:N (multiple publishers, topics, subscribers) Persistent test, conducted in binding mode.

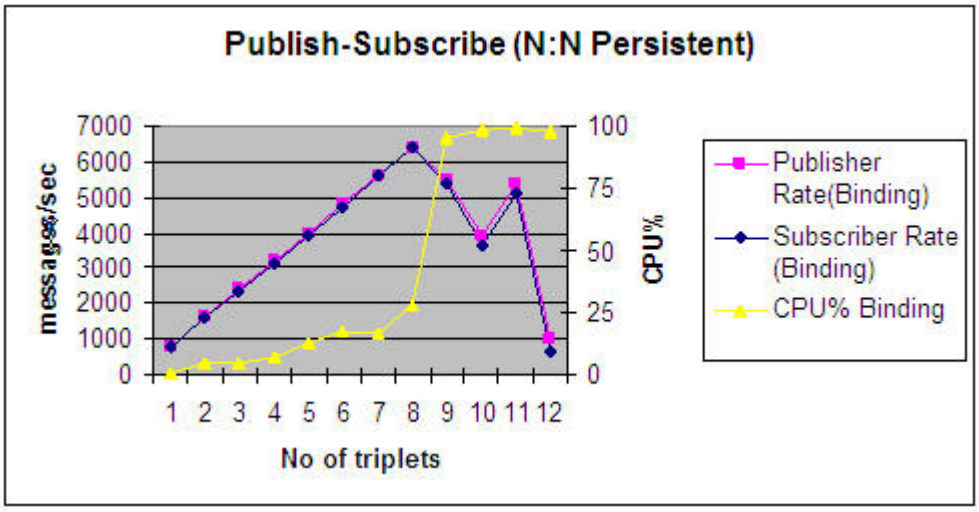

Figure 25. Publish Subscribe N:N Persistent, Binding chart

During the test, the publisher rate and subscriber rate linearly increase up to 8 triplets. The CPU utilization also increases linearly.

Managed Client mode

The graph in Figure 26, displays the results of Publish Subscribe N:N (multiple publishers, topics, subscribers) Persistent test, conducted in managed client mode.

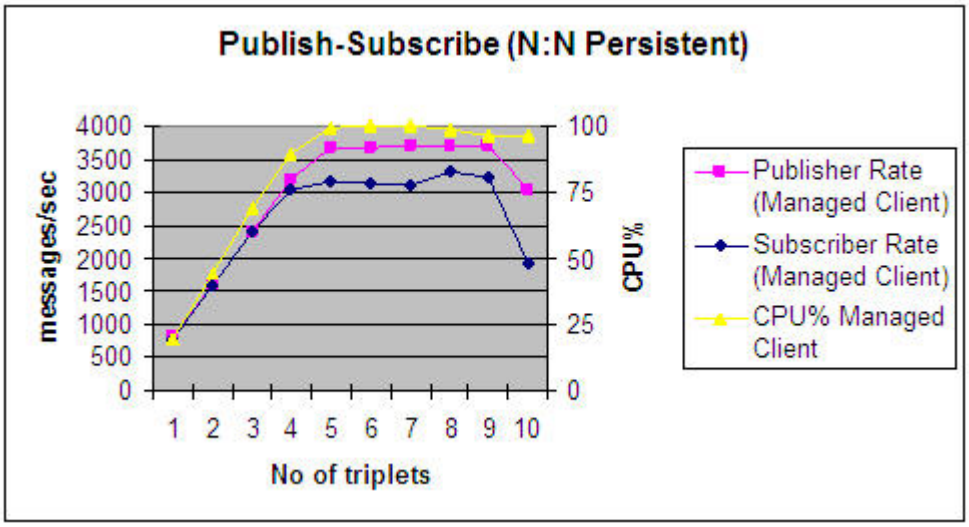

Figure 26. Publish Subscribe N:N Persistent, Managed Client chart

During the test, the publisher rate and subscriber rate increases linearly until the CPU utilization reaches maximum. After that, the performance remains constant with an increase in the number of applications.

Unmanaged Client Mode

The graph in Figure 27, displays the results of Publish Subscribe N:N (multiple publishers, topics, subscribers) Persistent test, conducted in unmanaged client mode.

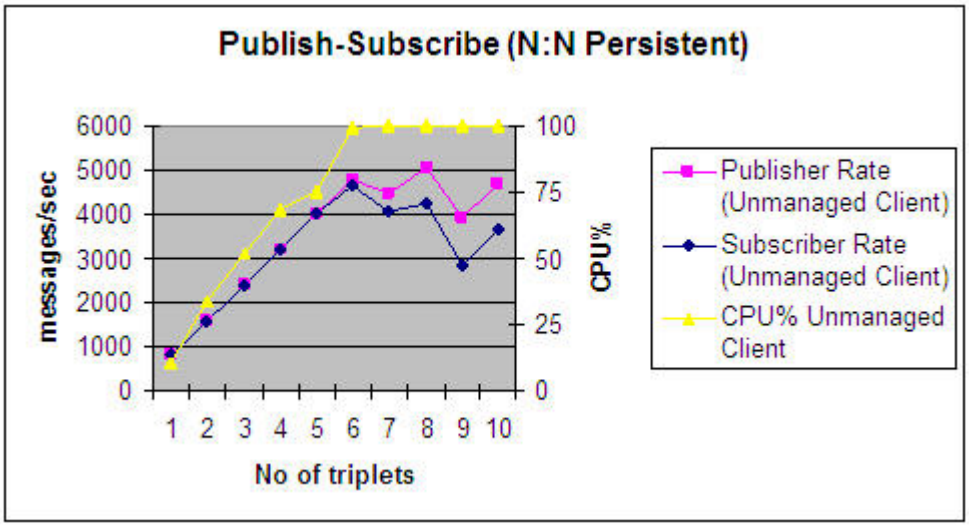

Figure 27. Publish Subscribe N:N Persistent, Unmanaged client chart

During the test, the publisher rate and subscriber rate increases linearly until the CPU utilization reaches maximum. After that, the performance remains constant with an increase in the number of applications.

When the subscriber is not able to keep up with the messages being published by the publisher, the messages are queued. Queuing indicates that superior resources are needed to increase the throughput.

The maximum performance statistics achieved with this configuration for persistent messages in binding, client, and unmanaged client modes is given in Table 11.

Table 11. Publish Subscribe N:N Persistent mode maximum rate

Persistent	**No of triplets**	**Publish rate in messages/ sec**	**Subscribe rate in messages/ sec**	**Expected total rate in messages/ sec**	**Actual total rate in messages/ sec**	**CPU%**
Binding	8	6399	6401	12800	12800	27.7
Managed Client	4	3200	3036	6400	6236	89.8
Unmanaged Client	6	4800	4652	9600	9452	99.6

The total rate is the sum of publisher rate and subscriber rate.

The total expected rate is calculated as:

Total expected rate = 800 * number of triplets * 2.

Chapter 5. Testing with WebSphere MQ V7 client features

This section provides the results of enabling WebSphere MQ V7 client features. Tests were run after enabling, and again after disabling, selected features of WebSphere MQ. The results are presented in charts and tables.

Performance comparison by varying ShareCnv

The graph displays the results of the Point to Point N:N (multiple producers, consumers, queues) Non-Persistent test, conducted in managed client mode by varying **ShareCnv**.

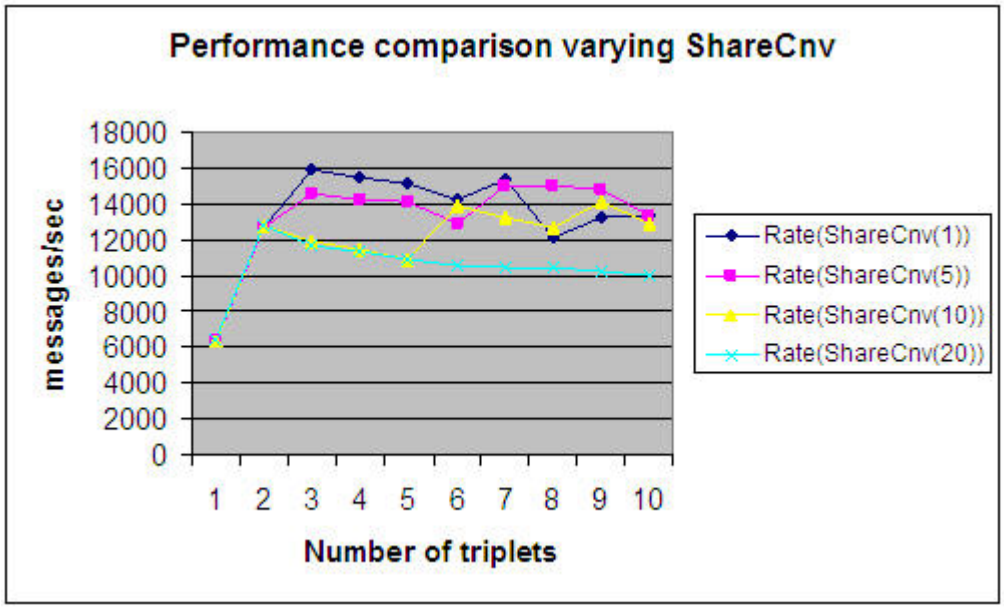

Figure 28. Performance comparison varying ShareCnv

The sender sends messages at a constant rate of 3200 messages per second. The number of senders, and receivers are gradually increased up to 10 triplets.

- The rate obtained is the sum of sender and receiver rate.
- The default **ShareCnv** value is 10. This means that, if an application opens more than one connection from a process, up to 10 connections share a single socket.
- The performance is measured in messages per second, after setting the **ShareCnv** value to 1, 5, 10, 20.

Table 12 shows the difference in performance by changing **ShareCnv** value when the number of triplets is 3.

Table 12. Performance Comparison by varying ShareCnv

Application Point-Point (N:N) NP, Managed Client mode	**ShareCnv(1)**	**ShareCnv(20)**
No of triplets	3	3
Rate	16000 messages/sec	11662 messages/sec

Performance comparison by using Asynchronous Put

The graph displays the results of Point-to-Point N:N (multiple producers, consumers, queues) Non-Persistent test, conducted in managed client mode with, and without Asynchronous Put enabled.

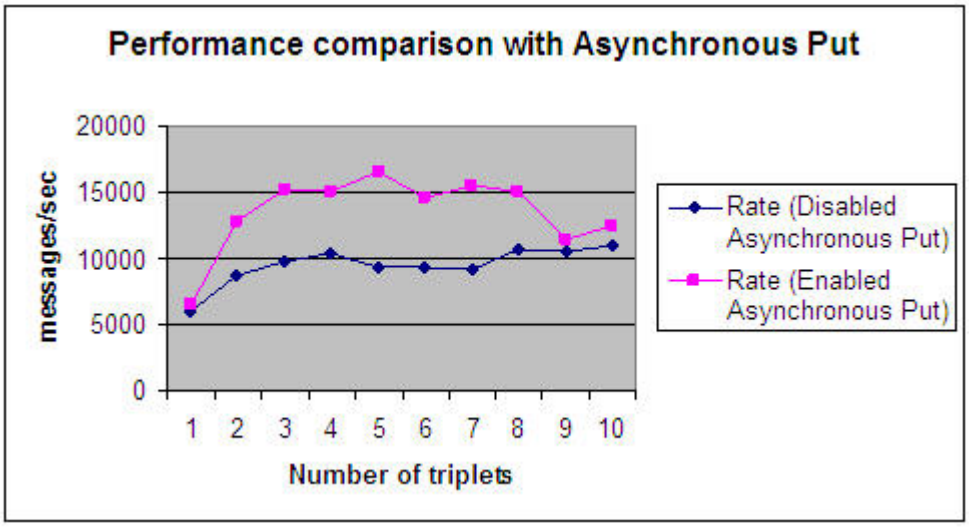

Figure 29. Performance comparison with Asynchronous Put

The sender sends messages at a constant rate of 3200 messages per second. The number of senders, receivers is gradually increased to 10 triplets.

- The rate obtained is the sum of the sender and receiver rate.
- The performance is measured with, and without Asynchronous Put enabled.
- The performance of sending application is higher when Asynchronous Put is enabled, as the sending application does not wait for acknowledgment for the sent messages from the queue manager.

Table 13 provides the resulting statistics when number of triplets is set to 5.

Table 13. Performance comparison by using Asynchronous Put

Application Point-Point (N:N) NP, Managed Client mode	**Read Ahead enabled**	**Without Read Ahead**
No of triplets	5	5
Rate	16594 messages/sec	9257 messages/sec

Performance comparison using Read Ahead

The Read Ahead feature is useful for non-persistent messages during receive.

The graph in Figure 30, displays the results of Point-to-Point N:N (multiple producers, consumers, queues) Non-Persistent test conducted in managed client mode with, and without Read Ahead enabled.

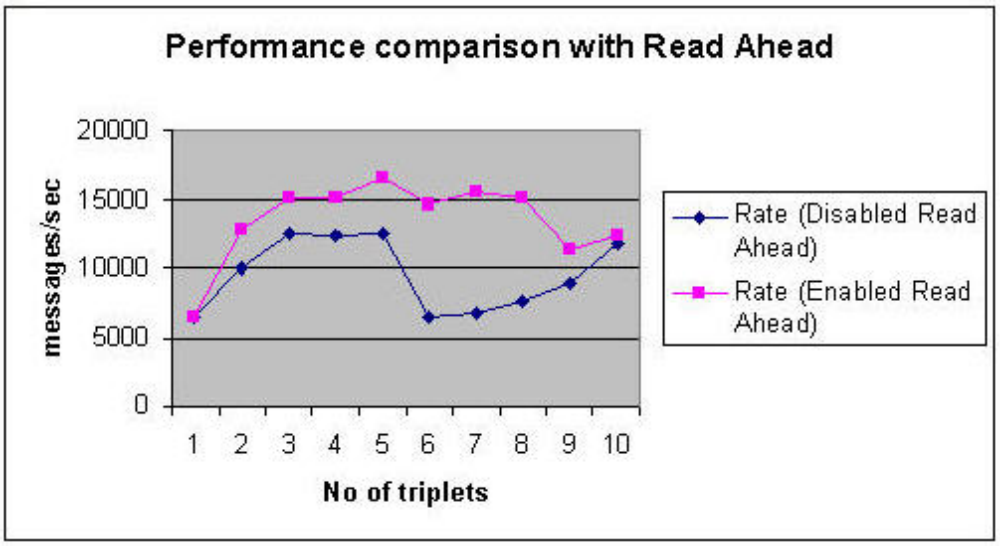

Figure 30. Performance comparison with Read Ahead

The Read Ahead buffer size is set to the default buffer size of 100k.

The sender sends messages at a constant rate of 3200 messages per second. The number of senders, receivers is gradually increased to 10 triplets.

- The rate obtained is the sum of the sender and receiver rate.
- The performance is measured with, and without Read Ahead enabled.
- The performance of the receiving application is higher when Read Ahead is enabled, as the receiving application reads the messages from queue manager without waiting for client to read the message.

The highest throughput for the application is with Read Ahead enabled. Statistics generated when the number of triplets set to 5, is shown in Table 14.

Table 14. Performance maximum enabling Read Ahead

Application Point-Point (N:N) NP, Managed Client mode	**Read Ahead enabled**	**Without Read Ahead**
No of triplets	5	5
Rate	16594 messages/sec	12558 messages/sec

Performance comparison by varying Read Ahead buffer Size

The graph in Figure 31 displays the results of Point-to-Point N:N (multiple producers, consumers, queues) Non-Persistent test, conducted in managed client mode with Read Ahead enabled through varying the Read Ahead buffer size.

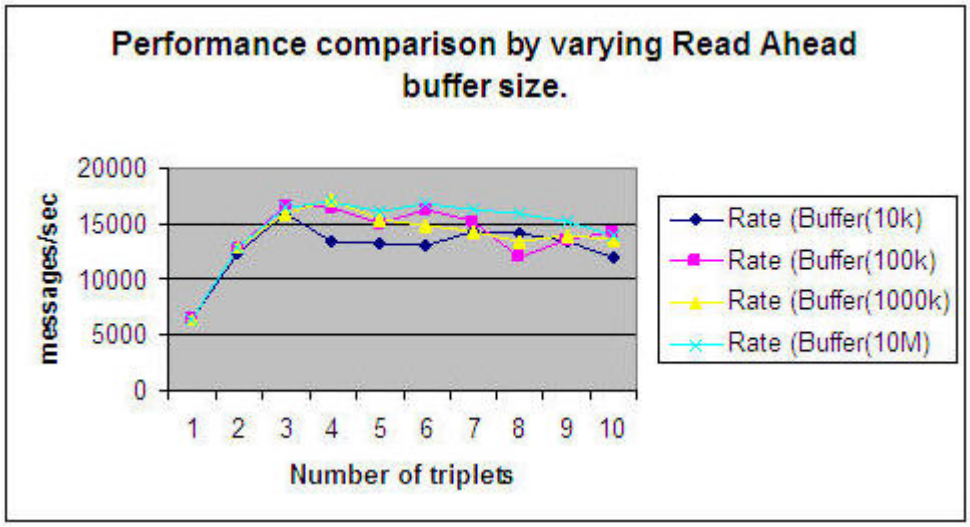

Figure 31. Performance comparison by varying Read Ahead buffer size

- The sender sends messages at a constant rate of 3200 messages per second. The number of senders, receivers is gradually increased to 10 triplets.
- The rate obtained is the sum of sender and receiver rate.
- The performance is measured by varying Read Ahead buffer size.
- The performance of the receiving application is higher if the Read Ahead buffer size is higher.

The highest throughput is for applications with Read Ahead enabled. Table 15 provides statistics generated when number of triplets is 4.

Table 15. Performance comparison by varying Read Ahead buffer size

Application Point-Point (N:N) NP, Managed Client mode	**Read Ahead buffer size 10 MB**	**Read Ahead buffer size 10 KB**
No of triplets	4	4
Rate	16887 messages/sec	13422 messages/sec

Chapter 6. Tuning and Programming guidelines

This section describes various tuning, and programming parameters that can be set to deliver better performance. Each section briefly describes the tuning parameter, and the settings used in the test environment.

Selecting the Log location

Select the WebSphere MQ log file location before creating the queue manager. It is preferable to have a fast (low write latency) local disk, or a SAN disk.

Create a directory `D:\MQM_LOG`

Use *–ld* option to set the log location. Create the queue manager using *crtmqm*.

```
crtmqm [-z] [-q] [-c Text] [-d DefXmitQ] [-h MaxHandles]
[-g ApplicationGroup]
[-t TrigInt] [-u DeadQ] [-x MaxUMsgs] [-lp LogPri] [-ls LogSec]
[-lc | -ll] [-lf LogFilePages] [-ld LogPath] QMgrName
```

Level of log write

The method to reliably write log records is selected by this variable. The default is **`TripleWrite`**.

For better performance, **`SingleWrite`** is preferred. **`SingleWrite`** can be selected if the disk attributes guarantee that all the written data will be written into the disk, in case of a failure.

Note: For reliability of data, **`TripleWrite`** is preferred when using non-cached disk.

Type of logging

WebSphere MQ provides two types of logging; linear and circular. In linear logging, the log extents are continuously allocated as required. In circular logging, the extents with inactive log data are reused.

The default is circular logging. For better performance, use circular logging.

To set the logging type, set the `-lc` option with **`crtmqm`**.

Log file extent size

The log data is held in a series of log files. The log file size is specified in units of 4-KB pages.

The default number of log file pages is 4096, giving a log file size of 16 MB on Windows.

Note: The minimum number of log file pages is 32, and the maximum is 65535. Allocate the maximum size, providing disk space is available.

To set the log file extent size, set the **`-lf`** option used with **`crtmqm`**.

Note: For performance testing, the maximum size of 65535 was used.

Number of log file extents

Log file extents can be specified as primary or secondary. Primary extents are allocated and formatted by the queue manager when it is first started, or when extra extents are added. Once a primary extent is formatted, it can be reused. Secondary log file extents are allocated dynamically by the queue manager when the primary files are exhausted.

The number of extents needed depends upon the amount of data to be logged, and the size of each extent.

The values used are: **LogPrimaryFiles**=10 and **LogSecondaryFiles**=15.

LogBufferPages

The Log Buffer is the amount of main memory used to accumulate log records that are written out to disk. Log records are appended at the end of the log buffer. When the end of the buffer is reached, serialization takes place. Serialization reduces the rate of data transfer to disk. A larger buffer reaches its limit less frequently than a smaller buffer.

You can specify the size of the buffer in units of 4Â-KB pages. Specify the value using the**LogBufferPages** parameter of the Log stanza of the queue manager (consider the platform on which the queue manager is located).

Note: Performance testing is done with **LogBufferPages** set to 4096.

Setting the Log parameter values in the registry

The log parameter values are set in the registry as follows.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\IBM\MQSeries\CurrentVersion\
Configuration\QueueManager\<Queue manager name>\Log]
"LogPrimaryFiles"="10"
"LogSecondaryFiles"="15"
"LogFilePages"="65535"
"LogType"="CIRCULAR"
"LogBufferPages"="4096"
"LogPath"="C:\\Program Files (x86)\\IBM\\WebSphere MQ\\log\\QM_xmspamdlnx2\\"
"LogWriteIntegrity"="SingleWrite"
```

Queue buffer sizes

Each queue in the queue manager is assigned with two buffers to hold messages. One buffer is for non-persistent messages. The other buffer is for persistent messages. When the messages exceed the size of the buffer, they are stored in the operating system file system. By increasing the size of the buffer limit, more messages can be stored in the buffer where they are readily available to the queue manager.

Defining queue buffers for queues (using large non-persistent or persistent messages) can degrade the performance if the system is in short of memory. This scenario might occur because many queues are defined with large buffers, or many channels. When increasing the values for the buffer sizes, ensure that increasing the size improves the message throughput. If throughput is not increased, then do not increase the buffer sizes.

When setting the values for buffer size, you must consider the average size and average number of messages that are on the queue. As with all queue-based processing in WebSphere MQ, aim to have low queue depths.

The non-persistent queue buffer size is specified using the tuning parameter **DefaultQBufferSize**. The persistent queue buffer size is specified using the tuning parameter **DefaultPQBufferSize**.

The queue buffer size values are set in registry as values for TuningParameters:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\IBM\MQSeries\
CurrentVersion\Configuration\
QueueManager\<Queue manager name>\TuningParameters]

"DefaultQBufferSize"="1048576"
"DefaultPQBufferSize"="1048576"
```

Queue manager channels

You can set **MQIBindType** to *FASTPATH* on the client channel.

The major benefit of this setting is performance improvement for non-persistent messages. This setting eliminates the AGENT process and reduces the processor (CPU) cost.

By specifying **MQIBindType**=*FASTPATH* in the registry, and setting the environment variable **MQ_CONNECT_TYPE**=*FASTPATH* this performance optimization is achieved.

Note: This setting is not applicable when using user exits, because the user code is run as part of the queue manager.

Enabling Prefetch in the system

The Prefetch feature of Windows operating system pre-fetches the data, anticipating cache misses. This feature improves the performance and is set in the registry.

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\
Memory Management\PrefetchParameters
EnablePrefetcher=3
```

Note: Performance can be improved by using different copy names of executable files for sender and receiver applications (if same executable file is used for sending and receiving).

Enabling Asynchronous Put

Using asynchronous put, an application can put a message to a queue without waiting for a response from the queue manager. This feature can be used to improve messaging performance in client applications.

```
Async Put is enabled on destination by setting XMSC.WMQ_PUT_ASYNC_ALLOWED
property as XMSC.WMQ_PUT_ASYNC_ALLOWED_ENABLED

Example:-
dest.SetIntProperty(XMSC.WMQ_PUT_ASYNC_ALLOWED, XMSC.WMQ_PUT_ASYNC_ALLOWED_ENABLED);
```

Note: For performance testing, **AsyncPut** is enabled.

Enabling Read Ahead

By enabling Read Ahead on a client receiver or subscriber, non-persistent messages can be streamed from queue manager to client without the client application having to request the messages.

When a client requires a message from a server, it sends a request to the server. It sends a separate request for each of the messages it consumes. To improve the performance of a client consuming non-persistent messages, a client can be configured to use Read Ahead. Read Ahead allows messages to be sent from a queue manager to a client without an application having to request them.

Read ahead is enabled on destination by setting on

```
 XMSC.WMQ_READ_AHEAD_ALLOWED
as XMSC.WMQ_READ_AHEAD_ALLOWED_ENABLED .
```

Example:

```
dest.SetIntProperty(XMSC.WMQ_READ_AHEAD_ALLOWED, XMSC.WMQ_READ_AHEAD_ALLOWED_ENABLED);
```

Note: For performance testing, Read Ahead is enabled.

Setting the ReadAhead buffer size

The **ReadAhead** buffer size is set by adding the **MessageBuffer** stanza to the `mqclient.ini` file.

For example, to add 1000-KB buffer size, perform the following step:

In the `mqclient.ini` file, add the **MessageBuffer** stanza:

```
MessageBuffer:
   MaximumSize=1000
   Updatepercentage=-1
   PurgeTime=0
```

Note: For performance testing, the default buffer size of 100-KB is used.

Disabling multiplexing and shared conversation

The **SHARECNV** channel attribute specifies the maximum number of conversations that share each TCP/IP channel instance. Sharing conversations has performance implications, as all the conversations on a socket are received by the same thread.

High **SHARECNV** limits have the advantage of reducing the queue manager thread usage. However, if many conversations sharing a socket are all busy, there is a possibility of delays as the conversations contend with one another to use the receiving thread. In this situation, a lower **SHARECNV** value is better.

Note: The default value on a client-connection channel for the **SHARECNV** channel attribute is 10.

Shared converstaion and multiplexing is disabled by setting

```
XMSC.WMQ_SHARE_CONV_ALLOWED on connection factory.
```

Example:

```
cf_.SetIntProperty(XMSC.WMQ_SHARE_CONV_ALLOWED, XMSC.WMQ_SHARE_CONV_ALLOWED_NO);
```

The number of connections to share is set on the channel by setting **SHARECNV** value.

Example:

```
alter CHANNEL(SYSTEM.DEF.SVRCONN)            CHLTYPE(SVRCONN) SHARECNV(1)
```

Note: For improved performance, shared conversation is disabled in the performance run configuration.

Send and Receive Buffer size for large messages

To change the buffer size, environment variables **MQ_COMMS_IP_RCVBUF**, and **MQ_COMMS_IP_SDRBUF** have to be configured. The default values are 32 KB.

For potential improved performance, set the **send** and **receive** buffer size to a higher value, the **readahead** buffer to the maximum value, and the **SHARECNV** channel attribute to 1.

Note: For performance testing, the default setting of 32 KB was used for the send and receive buffer size.

Chapter 7. Machine and Test Configuration

The following test configuration is required:

- Hardware
  Model: IBM® System x3655, 64-bit
  Type: 7985-41A
  4 CPU, Dual Core AMD Opteron processor 2216
  2.40 GHz, 8-GB RAM
- Disk
  Adaptec Array SCSI Disk
  Model: MBB2147RC
  Rotational Speed: 10K rpm, Buffer size 16 MB
  Transfer to Host: 3-GB/sec
  Average latency: 2.99 ms
- Operating System
  Microsoft Windows Server 2003 Enterprise x64 Edition, Version 5.2.3790, Service Pack 2
  NET Framework 2.0 Configuration, Version 2.0.50727.42
- WebSphere MQ
  Name: WebSphere MQ
  Version: 7.1.0.0
  Level: p000-L110128
  Mode: 32-bit
- XMS
  XMS (.NET) Version: 2.1.0.0
  CMVC Level: nn00-L110118

Chapter 8. Appendix

CPU Utilization

The percentage of time the processor was busy during the sampling interval. This counter is equivalent to the Task Manager's CPU Usage counter.

For the total processor utilization systemwide, the Processor(_Total)\% Processor Time counter is used.

The command used to get the total CPU utilization for the system is

```
typeperf -sc 1 "\processor(_total)\% processor time"
```

Average disk queue length

The Average disk queue length tracks the number of requests that are queued and waiting for a disk during the sample interval, as well as requests in service. As a result, this might overstate activity.

If more than two requests are continuously waiting on a single-disk system, the disk might be a bottleneck. To analyze queue length data further, use Avg. Disk Read Queue Length and Avg. Disk Write Queue Length.

The command used to get the average disk queue length is

```
typeperf -sc 1 "\PhysicalDisk(_Total)\Avg. Disk Queue Length"
```

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law. INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you. References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any references to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility. IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood New York 10594, USA. The information contained in this document has not be submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- AIX
- IBM
- WebSphere MQ

The following terms are trademarks of other companies:

- HP-UX Hewlett-Packard Development Company, L.P.
- Solaris Sun Microsystems, Inc.
- Windows Microsoft Corporation

Index

A

Architecture 5
Async put 33

B

Binding mode 17
Buffer size for large messages 35

H

Hardware 37

L

Log buffer pages 32
Log file Extent Size 31
Log file extents 32
Log location 31
Log Parameter values 32
Log write 31
Logging 31

M

Machine 37
Managed Client mode 18
Multiplexing 34

N

Non persistent results 11, 22
Non-persistent results 9

O

Operating system 37
Overview 3

P

Performance comparison by ShareCnv 27
Performance comparison by varying Read Ahead buffer size 30
Performance comparison using Asynchronous Put 28
Performance Comparison using Read Ahead 28
Performance enhancements 27
Performance in test scenarios 9
Performance test 5
Persistent results 10, 13, 24
Point to Point N:N (multiple producers, consumer, queues) scenario 6
Point to Point N:N (multiple producers, consumer) Scenario 11
Point to Point Put/Get 4Q scenario 5
Point to Point Put/Get 4Q Scenario 9
Prefetch 33
Programming guidelines 31
Publish Subscribe 1:N (one publisher, multiple subscribers) scenario 17
Publish Subscribe 1:N (Single publisher, multiple subscribers) scenario 7
Publish Subscribe N:N (multiple publishers, subscribers, topics) scenario 8
Publish Subscribe N:N (multiple publishers, subscribers) scenario 22

Q

Queue buffer sizes 32
Queue Manager channels 33

R

Read ahead 34
Read ahead buffer size 34

S

Shared conversation 34

T

Test configuration 37
Trademarks and service marks 43
Tuning guidelines 31

U

Unmanaged Client mode 20

Printed in USA