

WebSphere MQ for Windows64 V7.0 - Performance Evaluations

Version 1.2

Septembere 2009

Peter Toghil , Craig Stirling .

WebSphere MQ Performance

IBM UK Laboratories

Hursley Park

Winchester

Hampshire

SO21 2JN

Property of IBM

Please take Note!

Before using this report, please be sure to read the paragraphs on “disclaimers”, “warranty and liability exclusion”, “errors and omissions”, and the other general information paragraphs in the “Notices” section below.

Second Edition, September 2009.

This edition applies to *WebSphere MQ for Windows 64 edition V7* (and to all subsequent releases and modifications until otherwise indicated in new editions).

© Copyright International Business Machines Corporation 2009. All rights reserved.

Note to U.S. Government Users

Documentation related to restricted rights.

Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

Notices

DISCLAIMERS

The performance data contained in this report were measured in a controlled environment. Results obtained in other environments may vary significantly.

You should not assume that the information contained in this report has been submitted to any formal testing by IBM.

Any use of this information and implementation of any of the techniques are the responsibility of the licensed user. Much depends on the ability of the licensed user to evaluate the data and to project the results into their own operational environment.

WARRANTY AND LIABILITY EXCLUSION

The following paragraph does not apply to the United Kingdom or any other 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 NON-INFRINGEMENT, 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.

In Germany and Austria, notwithstanding the above exclusions, IBM's warranty and liability are governed only by the respective terms applicable for Germany and Austria in the corresponding IBM program license agreement(s).

ERRORS AND OMISSIONS

The information set forth in this report could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; any such change will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time and without notice.

INTENDED AUDIENCE

This report is intended for architects, systems programmers, analysts and programmers

wanting to understand the performance characteristics of *WebSphere MQ for Windows64 V7*. The information is not intended as the specification of any programming interface that is provided by WebSphere. It is assumed that the reader is familiar with the concepts and operation of WebSphere MQ V7.

LOCAL AVAILABILITY

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates. Consult your local IBM representative for information on the products and services currently available in your area.

ALTERNATIVE PRODUCTS AND SERVICES

Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

USE OF INFORMATION PROVIDED BY YOU

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

TRADEMARKS AND SERVICE MARKS

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

- IBM
- WebSphere
- DB2

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

EXPORT REGULATIONS

You agree to comply with all applicable export and import laws and regulations.

Preface

Target audience

This SupportPac is designed for people who:

- Will be designing and implementing solutions using WebSphere MQ for Windows64.
- Want to understand the performance limits of WebSphere MQ for Windows64 V7.0.
- Want to understand what actions may be taken to tune WebSphere MQ for Windows64.

The reader should have a general awareness of the Windows64 operating system and of MQSeries in order to make best use of this SupportPac. Readers should read the section '**How this document is arranged**'—**Page VI** to familiarise themselves with where specific information can be found for later reference.

The contents of this SupportPac

This SupportPac includes:

- Release highlights performance charts.
- Performance measurements with figures and tables to present the performance capabilities of WebSphere MQ local queue manager, client channel, and distributed queuing scenarios.
- Interpretation of the results and implications on designing or sizing of the WebSphere MQ local queue manager, client channel, and distributed queuing configurations.

Feedback on this SupportPac

We welcome constructive feedback on this report.

- Does it provide the sort of information you want?
- Do you feel something important is missing?
- Is there too much technical detail, or not enough?
- Could the material be presented in a more useful manner?

Please direct any comments of this nature to **WMQPG@uk.ibm.com**.

Specific queries about performance problems on your WebSphere MQ system should be directed to your local IBM Representative or Support Centre.

Acknowledgements

Introduction

The three scenarios used in this report to generate the performance data are:

- Local queue manager scenario.
- Client channel scenario.
- Distributed queuing scenario.

Unless otherwise specified, the standard message sized used for all the measurements in this report is 2K (2,048 bytes).

An xSeries 3850 4 cpu 3.3ghz Intel xeon with 4GB of RAM was used as the Device under test.

An xSeries 3850 4 cpu 3.33ghz Intel xeon with 4GB of RAM was used as the Driver

How this document is arranged

Performance Headlines

Pages: 1-13

Section one contains the performance *headlines* for each of the three scenarios, with MQI applications connected to:

- A local queue manager.
- A remote queue manager over MQI-client channels.
- A local queue manager, driving throughput between the local and remote queue manager over server channel pairs.

The headline tests show:

- The maximum message throughput achieved with an increasing number of MQI applications.
- The maximum number of MQI-clients connected to a queue manager.
- The maximum number of server channel pairs between two queue managers, for a fixed think time between messages until the response time exceeds one second.

Large Messages

Pages: 18-38

Section two contains performance measurements for *large messages*. This includes *MQI response times* of 50byte to 2MB messages. It also includes *20K*, *200k* and *2M* messages using the same scenarios as for the “*Performance Headlines*”.

Application Bindings

Page: 39-44

Section three contains performance measurements for ‘*trusted, normal, and isolated*’ server applications, using the same three scenarios as for the “*Performance Headlines*”.

Capacity Measurements

Pages: 46-

Are designed to be useful for system sizing

Measurement Environment

Pages: 53 54

A summary of the way in which the workload is used in each test scenario is given in the “*Performance Headlines*” section. This includes a more detailed description of the workload, hardware and software specifications.

Glossary

Page: 54

A short glossary of the terms used in the tables throughout this document.

CONTENTS

1	Overview	1
2	Performance Headlines	2
2.1	Local Queue Manager Test Scenario.....	2
2.1.1	Nonpersistent Messages – Local Queue Manager.....	3
2.1.2	Nonpersistent Messages – Non trusted – Local Queue Manager.....	4
2.1.3	Persistent Messages – Local Queue Manager.....	5
2.2	Client Channels Test Scenario.....	6
2.2.1	Nonpersistent Messages – Client Channels.....	7
2.2.2	Nonpersistent Messages – Non Trusted Client Channels.....	8
2.2.3	Persistent Messages – Client Channels.....	9
2.2.4	Client Channels.....	10
2.3	Distributed Queuing Test Scenario.....	12
2.3.1	Nonpersistent Messages – Server Channels.....	13
2.3.2	Non Persistent non Trusted – Server Channels.....	14
2.3.3	Persistent Messages – Server Channels.....	15
2.3.4	Server Channels.....	16
3	Large Messages	18
3.1	MQI Response Times: 50bytes to 100MB – Local Queue Manager.....	18
3.1.1	50bytes to 32KB.....	18
3.1.2	32KB to 2MB.....	19
3.1.3	2MB to 100MB.....	20
3.2	20K Messages.....	21
3.2.1	Local Queue Manager.....	21
3.2.2	Client Channel.....	23
3.2.3	Distributed Queuing.....	25
3.3	200K Messages.....	27
3.3.1	Local Queue Manager.....	27
3.3.2	Client Channel.....	29
3.3.3	Distributed Queuing.....	31
3.4	2MB Messages.....	33
3.4.1	Local Queue Manager.....	33
3.4.2	Client Channel.....	35
3.4.3	Distributed Queuing.....	37
4	Application Bindings	39
4.1	Local Queue Manager.....	39
4.1.1	Nonpersistent Messages.....	39
4.1.2	Persistent Messages.....	40
4.2	Client Channels.....	41
4.2.1	Nonpersistent Messages.....	41
4.2.2	Persistent Messages.....	42
4.3	Distributed Queuing.....	43
4.3.1	Nonpersistent Messages.....	43
4.3.2	Persistent Messages.....	44
5	Short & Long Sessions	45
6	Performance and Capacity Limits	46
6.1	Client channels – capacity measurements.....	46
6.2	Distributed queuing – capacity measurements.....	47
7	Tuning Recommendations	48
7.1	Tuning the Queue Manager.....	48
7.1.1	Queue Disk, Log Disk, and Message Persistence.....	48
7.1.2	Log Buffer Size, Log File Size, and Number of Log Extents.....	48
7.1.3	Channels: Process or Thread, Standard or Fastpath?.....	50
7.1.4	Multiplexed clients.....	50
7.2	Applications: Design and Configuration.....	50
7.2.1	<i>Standard (Shared or Isolated)</i> or Fastpath?.....	50
7.2.2	Parallelism, Batching, and Triggering.....	50
7.3	Virtual Memory, Real Memory, & Paging.....	51
7.3.1	Queue Manager.....	51
7.3.2	Channels.....	51
7.3.3	Client Channels.....	51
7.3.4	Server – Server Channels.....	52

7.3.5	Reply Queue.....	52
7.3.6	BufferLength.....	52
7.3.7	MQIBINDTYPE.....	52
7.4	TCP/IP tuning for capacity number.....	52
8	Measurement Environment.....	53
8.1	Workload description.....	53
8.1.1	MQI response time tool.....	53
8.1.2	Test scenario workload.....	53
8.2	Hardware.....	54
8.3	Software.....	54
9	Glossary.....	55

TABLES

Table 1 – Performance headline, nonpersistent messages, local queue manager	3
Table 2 – Performance headline, persistent messages, local queue manager	5
Table 3 – Performance headline, nonpersistent messages, client channels	7
Table 4 – Performance headline, persistent messages, client channels	9
Table 5 – 1 round trip per driving application per second, client channels	11
Table 6 – Performance headline, nonpersistent messages, server channels	13
Table 7 – Performance headline, persistent messages, server channels	15
Table 8 – 1 round trip per driving application per second, client channels	17
Table 9 – 20K nonpersistent messages, local queue manager	21
Table 10 – 20K persistent messages, local queue manager	22
Table 11 – 20K nonpersistent messages, client channels	23
Table 12 – 20K persistent messages, client channels	24
Table 13 – 20K nonpersistent messages, client channels	25
Table 14 – 20K persistent messages, client channels	26
Table 15 – 200K nonpersistent messages, local queue manager	27
Table 16 – 200K persistent messages, local queue manager	28
Table 17 – 200K nonpersistent messages, client channels	29
Table 18 – 200K persistent messages, client channels	30
Table 19 – 200K nonpersistent messages, distributed queuing	31
Table 20 – 200K persistent messages, distributed queuing	32
Table 21 – 2M nonpersistent messages, local queue manager	33
Table 22 – 2M persistent messages, local queue manager	34
Table 23 – 2M nonpersistent messages, client channels.....	35
Table 24 – 2M persistent messages, client channels	36
Table 25 – 2M nonpersistent messages, distributed queuing	37
Table 26 – 2M persistent messages, distributed queuing	38
Table 27 – Application binding, nonpersistent messages, local queue manager	39
Table 28 – Application binding, persistent messages, local queue manager	40
Table 29 – Application binding, nonpersistent messages, client channels	41
Table 30 – Application binding, persistent messages, client channels	42
Table 31 – Application binding, nonpersistent messages, distributed queuing	43
Table 32 – Application binding, persistent messages, distributed queuing	44
Table 33 – Short sessions, client channels.....	45
Table 34 Client Capacity, throughput.....	46
Table 35 – Client capacity, memory utilisation.....	46
Table 36 – Capacity measurements, server channels	47
Table 37 – DQ capacity, memory utilisation.....	47

FIGURES

Figure 1 – Connections into a local queue manager	2
Figure 2 – Performance headline, nonpersistent messages, local queue manager	3
Figure 3 – Performance headline, persistent messages, local queue manager	5
Figure 4 – MQI-client channels into a remote queue manager	6
Figure 5 – Performance headline, nonpersistent messages, client channels	7
Figure 6 – Performance headline, persistent messages, client channels	9
Figure 7 – 1 round trip per driving application per second, client channels, nonpersistent messages	10
Figure 8 – 1 round trip per driving application per second, client channels, persistent messages	11
Figure 9 – Server channels between two queue managers	12
Figure 10 – Performance headline, nonpersistent messages, server channels	13
Figure 11 – Performance headline, persistent messages, server channels	15
Figure 12 – 1 round trip per driving application per second, server channel, nonpersistent messages ..	16
Figure 13 – 1 round trip per driving application per second, server channel, persistent messages	16
Figure 14 –The effect of nonpersistent message size on MQI response time (50byte - 32K)	18
Figure 15 –The effect of persistent message size on MQI response time (50byte - 32K)	18
Figure 16 –The effect of nonpersistent message size on MQI response time (32K – 2MB)	19
Figure 17 –The effect of persistent message size on MQI response time (32K – 2MB)	19
Figure 18 –The effect of nonpersistent message size on MQI response time (2MB – 100MB)	20
Figure 19 –The effect of persistent message size on MQI response time (2MB – 100MB)	20
Figure 20 – 20K nonpersistent messages, local queue manager	21
Figure 21 – 20K persistent messages, local queue manager	22
Figure 22 – 20K nonpersistent messages, client channels	23
Figure 23 – 20K persistent messages, client channels	24
Figure 24 – 20K nonpersistent messages, distributed queuing	25
Figure 25 – 20K persistent messages, distributed queuing	26
Figure 26 – 200K nonpersistent messages, local queue manager	27
Figure 27 – 200K persistent messages, local queue manager	28
Figure 28 – 200K nonpersistent messages, client channels	29
Figure 29 – 200K persistent messages, client channels	30
Figure 30 – 200K nonpersistent messages, distributed queuing	31
Figure 31 – 200K persistent messages, distributed queuing	32
Figure 32 – 2M nonpersistent messages, local queue manager	33
Figure 33 – 2M persistent messages, local queue manager	34
Figure 34 – 2M nonpersistent messages, client channels	35
Figure 35 – 2M persistent messages, client channels	36
Figure 36 – 2M nonpersistent messages, distributed queuing	37
Figure 37 – 2M persistent messages, distributed queuing	38
Figure 38 – Application binding, nonpersistent messages, local queue manager	39
Figure 39 – Application binding, persistent messages, local queue manager	40
Figure 40 – Application binding, nonpersistent messages, client channels	41
Figure 41 – Application binding, persistent messages, client channels	42
Figure 42 – Application binding, nonpersistent messages, distributed queuing	43
Figure 43 – Application binding, persistent messages, distributed queuing	44

1 Overview

WebSphere MQ V7.0 on Windows64 has similar performance characteristics to the V6. The comparisons in this report show that throughput is similar (for Local, Client and Distributed Queuing) when the Clients are running in V6 compatibility mode (see section 7.1.4). The default enhanced client support that provides Heartbeating, enhanced reliability, and multiplexing degrades Client benchmarks by a further 15%.

There are new functions in V7 that provide enhanced performance to applications that are able to use them and they include Asynchronous Puts, Read-ahead, Properties, and selectors but they are not covered in this document.

2 Performance Headlines

The measurements for the local queue manager scenario are for processing messages with no *think-time*. For the client channel scenario and distributed queuing scenario, there are also measurements for *rated* messaging.

No think-time is when the driving applications do not wait after getting a reply message before submitting subsequent request messages—this is also referred to as *tight-loop*.

The rated messaging tests used one round trip per driving application per *second*. In the client channel test scenarios, each driving application using a dedicated MQI-client channel, in the distributed queuing test scenarios, one or more applications submit messages over a fixed number of server channels.

All tests are automatically stopped after the response time exceeds 1 second.

2.1 Local Queue Manager Test Scenario

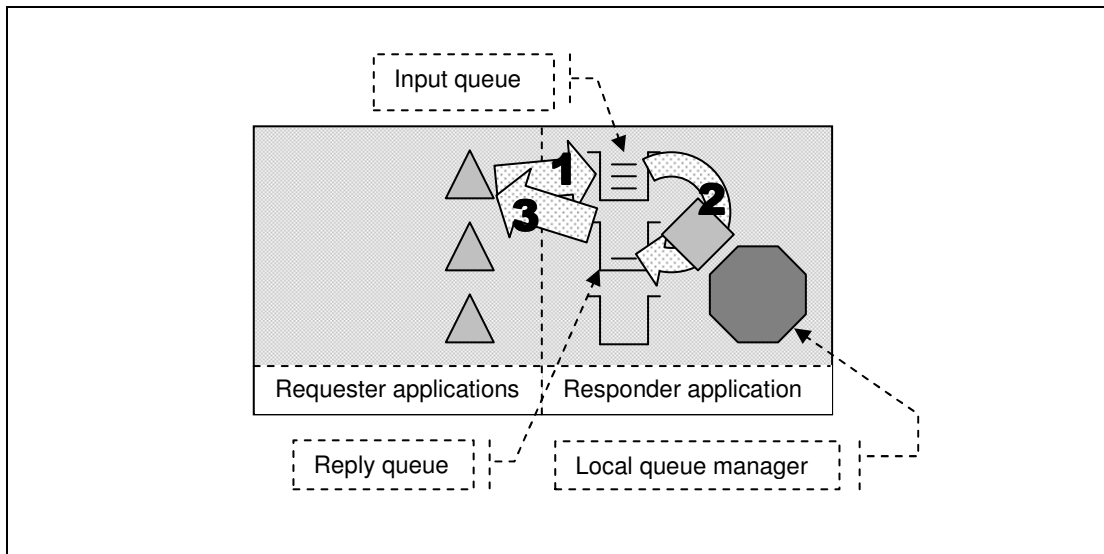


Figure 1 – Connections into a local queue manager

- 1) The Requester application puts a message to the common input queue on the local queue manager, and holds on to the message identifier returned in the message descriptor. The Requester application then waits indefinitely for a reply to arrive on the common reply queue.
- 2) The Responder application gets messages from the common input queue and places a reply to the common reply queue. The queue manager copies over the message identifier from the request message to the correlation identifier of the reply message.
- 3) The Requester application gets a reply from the common reply queue using the message identifier held from when the request message was put to the common input queue, as the correlation identifier in the message descriptor.

Nonpersistent and persistent messages were used in the local queue manager tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in the “*Large Messages*” section.

Application Bindings of the Responder program are ‘Shared’ and the Requester program is normally ‘Trusted’ except in the ‘non-trusted’ scenario where both programs use ‘shared’ bindings.

2.1.1 Nonpersistent Messages – Local Queue Manager

Figure 2 , Figure 2a , and Figure 3 shows the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario (see Figure 1 on the previous page), and WebSphere MQ V7.0 compared to Version 6.0.

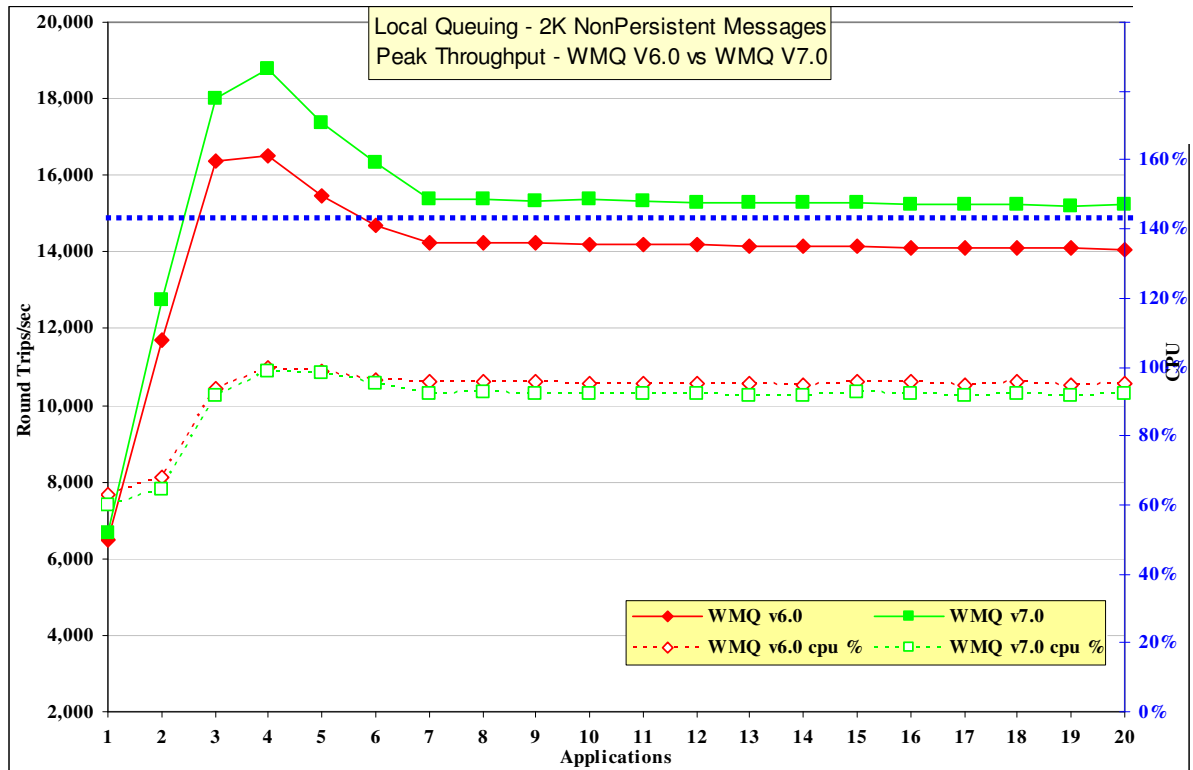


Figure 2 – Performance headline, nonpersistent messages, local queue manager

Note: Messaging in these tests is with no think-time.

Figure 2 and Table 1 shows that the throughput of nonpersistent messages has increased by 8% comparing Version 6 to Version 7

Test name: local_np	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	4	16523	0.0002	99%
WebSphere MQ V7.0	4	18754	0.0002	99%

Table 1 – Performance headline, nonpersistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

2.1.2 Nonpersistent Messages – Non trusted – Local Queue Manager

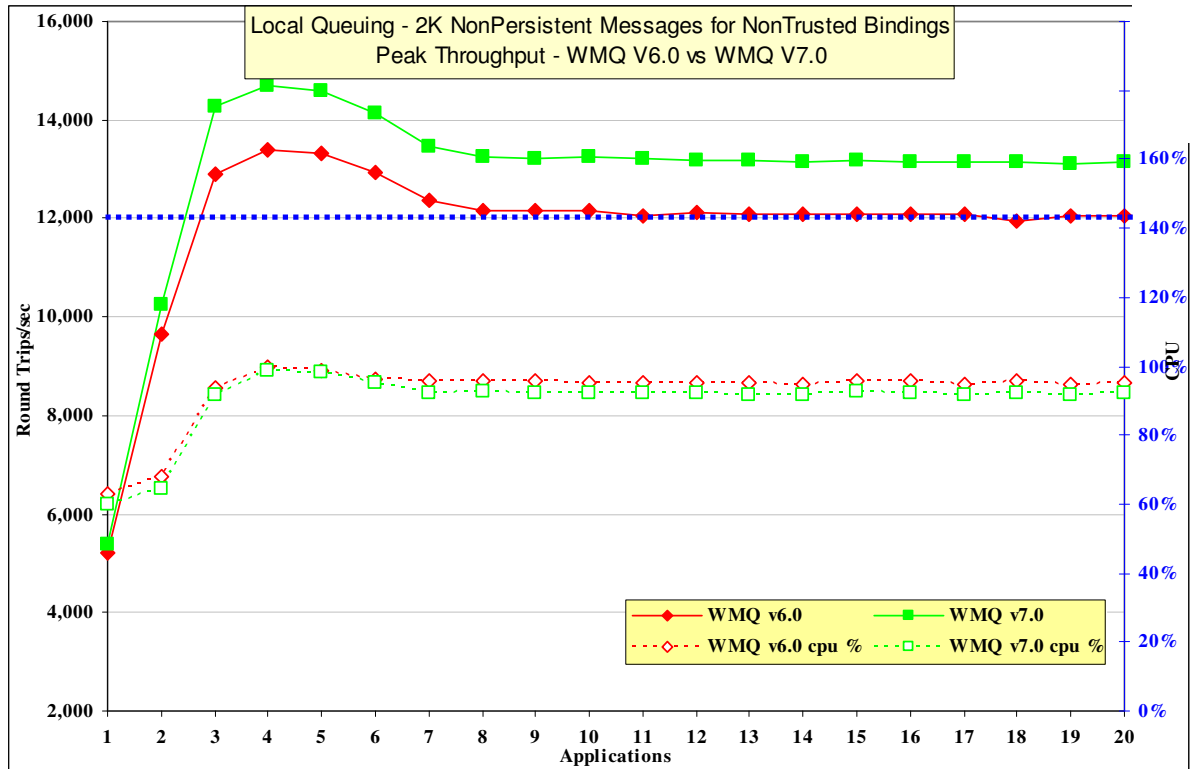


Figure 2a

Test name: local_np_nt	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	4	13390	0.0002	100%
WebSphere MQ V7.0	4	14680	0.0003	100%

shows that the throughput of nonpersistent messages when the Requester and Responder both use Shared bindings has increased by 8% comparing Version 6 to Version 7.

2.1.3 Persistent Messages – Local Queue Manager

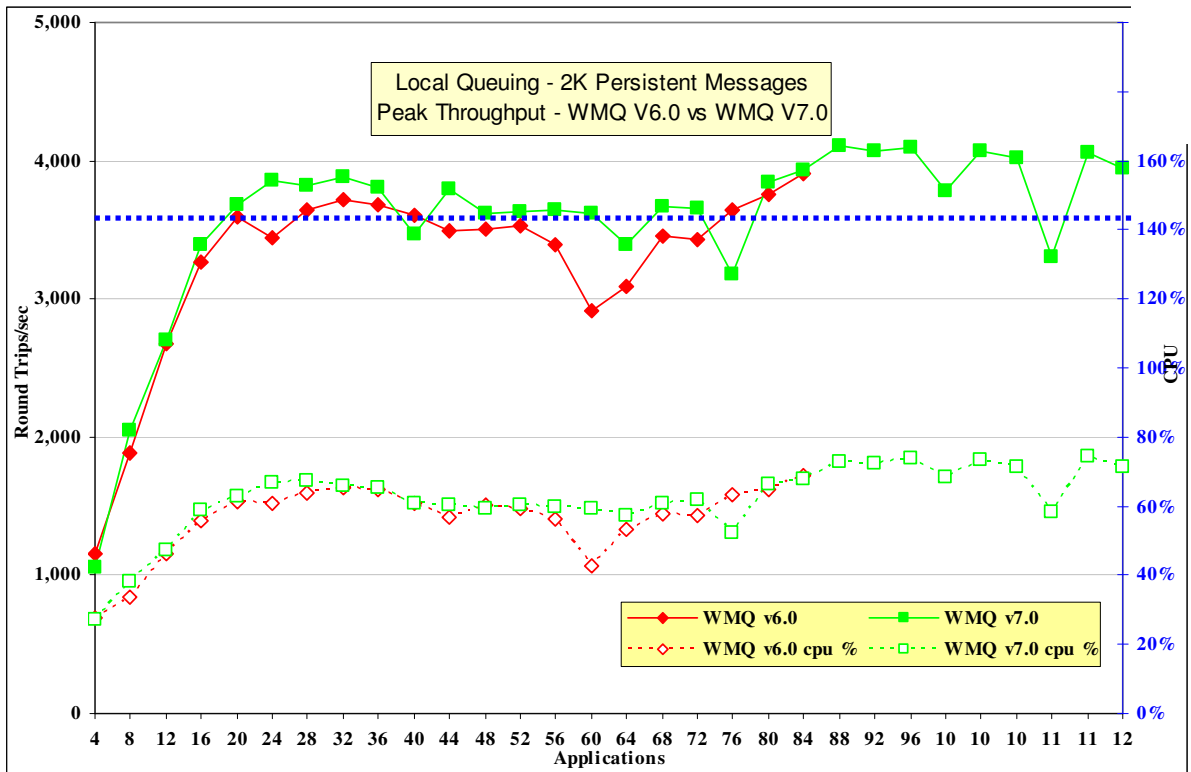


Figure 3 – Performance headline, persistent messages, local queue manager

Note: Messaging in these tests is with no think-time.

Figure 3 and Table 2 show that the throughput of persistent messages has increased by 4% when comparing Version 6.0 to Version 7.0.

Test name: local_pm	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	84	3907	0.024	68%
WebSphere MQ V7.0	(84) 88	(3937) 4106	(0.024) 0.026	(66%) 70%

Table 2 – Performance headline, persistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

2.2 Client Channels Test Scenario

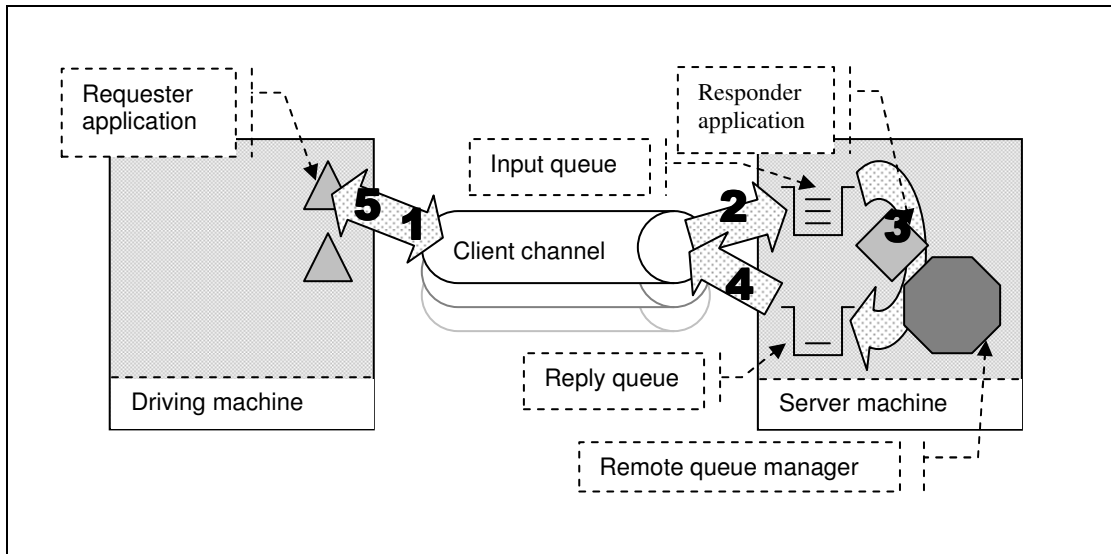


Figure 4 – MQI-client channels into a remote queue manager

- 1, 2) The Requester application puts a request message (over a client channel), to the common input queue, and holds on to the message identifier returned in the message descriptor. The Requester application then waits indefinitely for a reply to arrive on the common reply queue.
- 3) The Responder application gets messages from the common input queue and places a reply to the common reply queue. The queue manager copies over the message identifier from the request message to the correlation identifier of the reply message.
- 4, 5) The Requester application gets the reply message (over the client channel), from the common reply queue. The Requester application uses the message identifier held from when the request message was put to the common input queue, as the correlation identifier in the message descriptor.

Nonpersistent and persistent messages were used in the client channel tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in the “*Large Messages*” section.

Application Bindings of the Responder program are ‘Shared’ and the Client Channel is set to ‘MQIBindType = FASTPATH’ except in the ‘non-trusted’ scenario where ‘MQIBindType = NORMAL’ is used.

Version 7 will multiplex multiple clients over one TCP socket. The version 6 behavior where each client had its own TCP socket can be set by specifying `Sharecnv(0)` on the client channel definition and is shown in the charts as ‘optimized’. Further information in section 7.1.4

2.2.1 Nonpersistent Messages – Client Channels

Figure 5, Figure 5a, and Figure 6 shows the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario (see Figure 4 on the previous page), and WebSphere MQ V7.0 compared to Version 6.0.

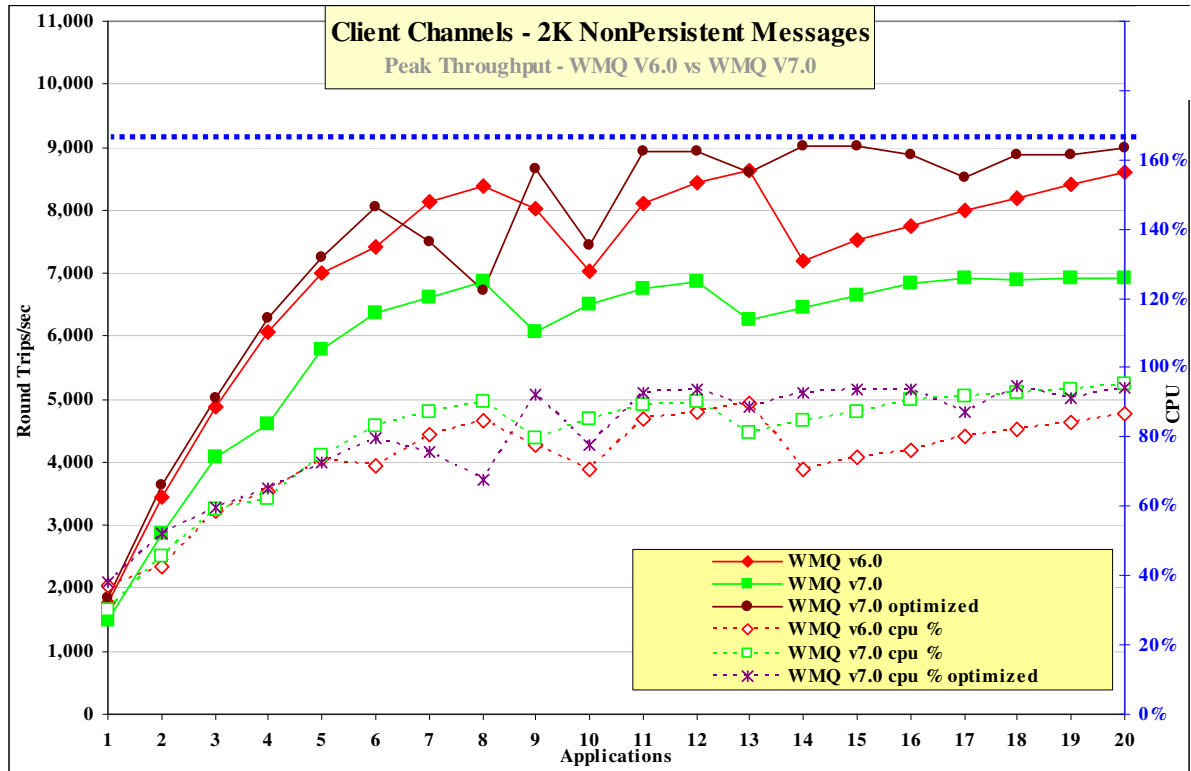


Figure 5 – Performance headline, nonpersistent messages, client channels

Note: Messaging in these tests is with no think-time

Figure 5 and Table 3 show that the throughput of nonpersistent messages has increased by 5% with optimised setup and degraded by 14% with default setup when comparing Version 6.0 to Version 7.0.

Test name: cInp	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	13	8636	0.002	72%
WebSphere MQ V7.0	(13)	(8607)	(0.003)	(93%)
Optimised	14	9018	0.003	94%
WebSphere MQ V7.0	(13)	(6265)	(0.003)	(81%)
	20	6930	0.004	95%

Table 3 – Performance headline, nonpersistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

2.2.2 Nonpersistent Messages – Non Trusted Client Channels

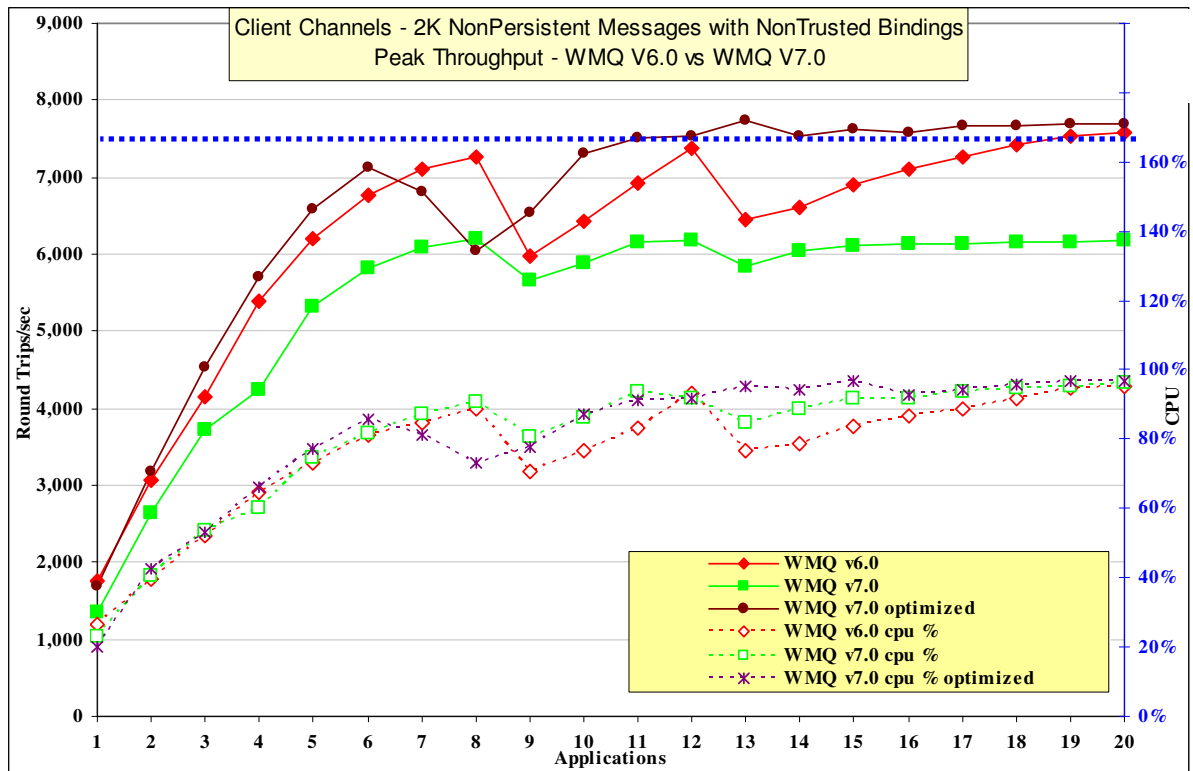


Figure 5a

Test name: Clnp_nt	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	20	7589	0.005	95%
WebSphere MQ V7.0 Optimised	13 (20)	7729 (7696)	0.002 (0.004)	96% (97%)
WebSphere MQ V7.0	8 (20)	6210 (6182)	0.002 (0.004)	93% (97%)

The throughput of nonpersistent messages when the channel has used the default MQIBINDTYPE=NORMAL has increased by 5% with optimised setup and degraded by 15% with default setup when comparing Version 6.0 to Version 7.0.

2.2.3 Persistent Messages – Client Channels

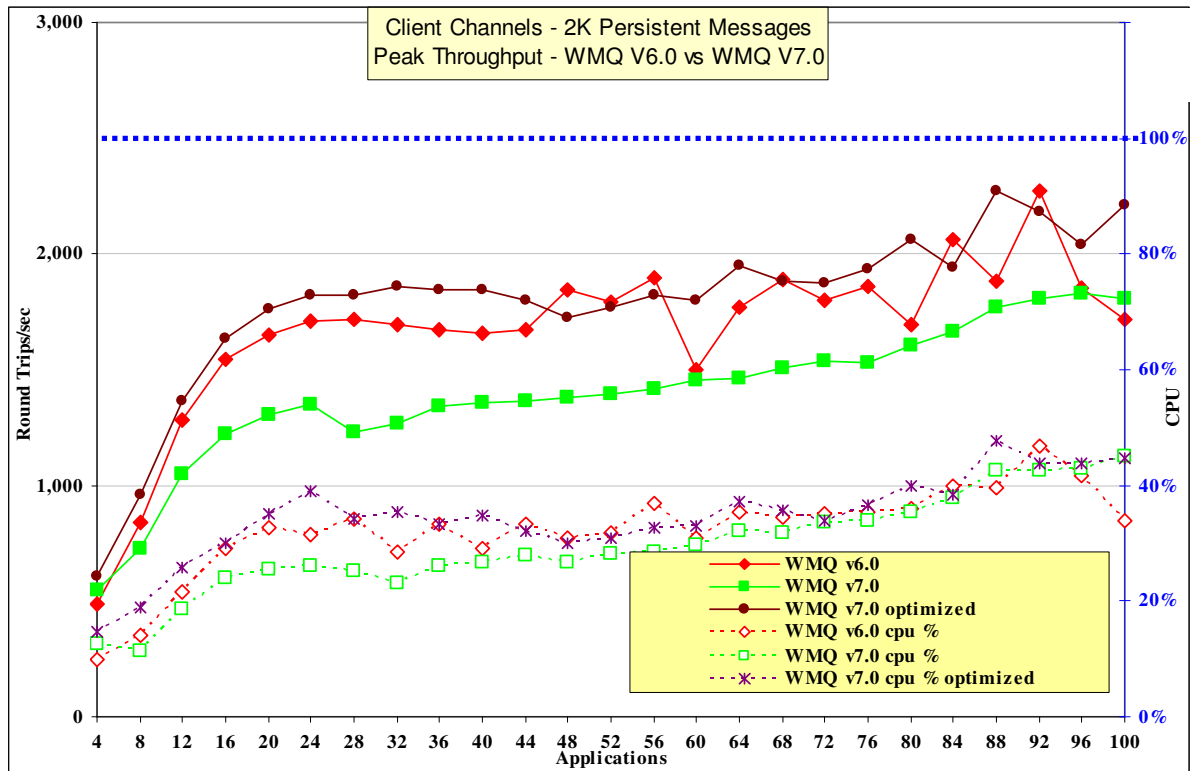


Figure 6 – Performance headline, persistent messages, client channels

Note: Messaging in these tests is with no think-time.

Figure 6 and Table 4 show that the throughput of persistent messages has decreased by 15% when using optimized setup and degraded by 15% using using default setup when comparing Version 6 to Version 7.

Test name: Cipm	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	92	2276	0.052	47%
WebSphere MQ V7.0 Optimised	(92) 100	(2182) 2215	(0.041) 0.062	(92%) 88%
WebSphere MQ V7.0	(92) 96	(1808) 1829	(0.051) 0.053	(92%) 96%

Table 4 – Performance headline, persistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

2.2.4 Client Channels

For the following client channel measurements, the messaging rate used is 1 round trip per *second* per MQI-client channel, i.e. a request message outbound over the client channel and a reply message inbound over the channel per second.

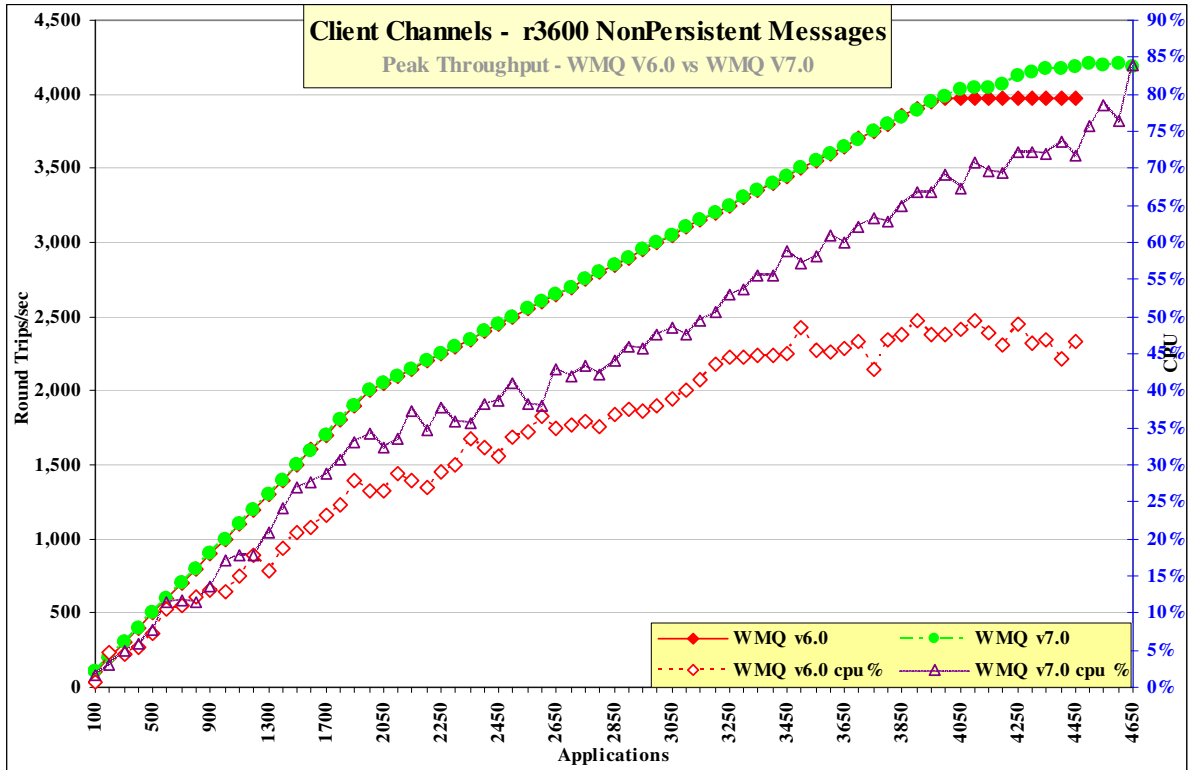


Figure 7 – 1 round trip per driving application per second, client channels, nonpersistent messages

Note: Messaging in these tests is 1 round trip per driving application per second.

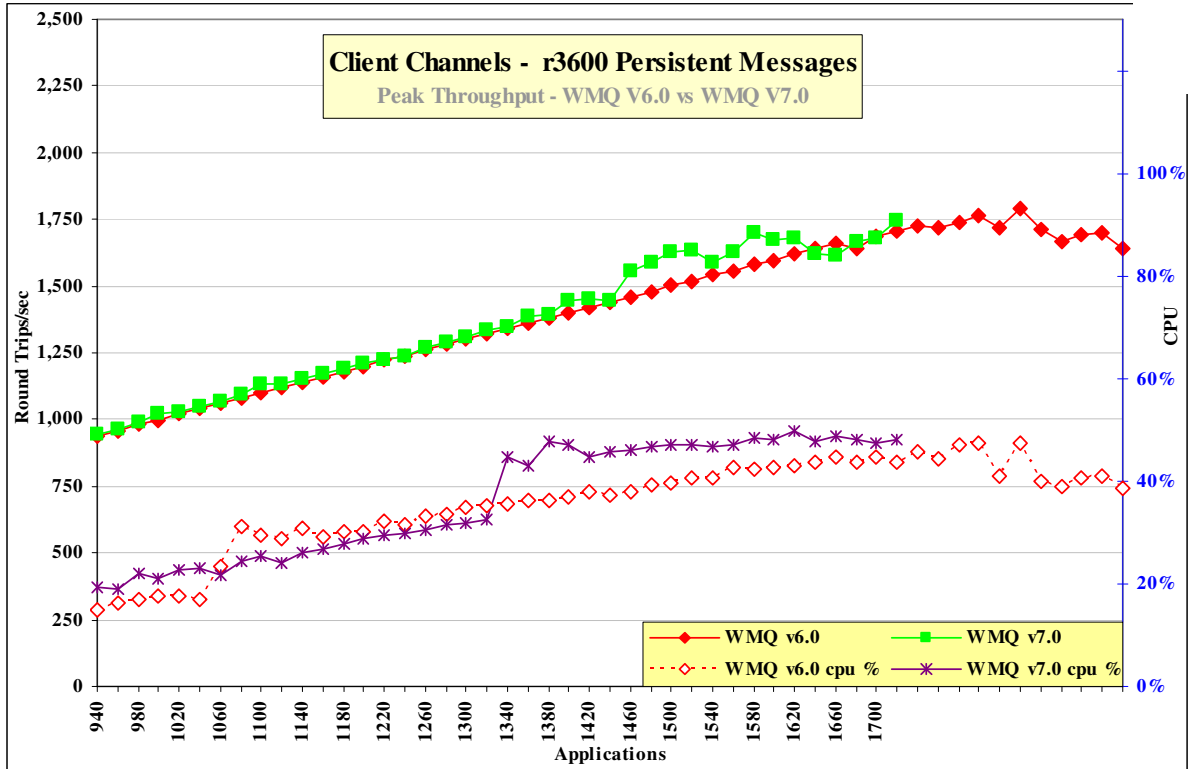


Figure 8 – 1 round trip per driving application per second, client channels, persistent messages

Figure 7, Figure 8 and Table 5 shows that WebSphere MQ V7.0 has similar maximum throughput with WebSphere MQ V 6.0.

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
clnp_r3600	4000	3,600	3987	0.280	69%
(WebSphereMQ v6.0)	(4000)		4173 (3970)	0.673 (0.049)	72% (48%)
clpm_r3600	1720	3,600	1743	0.019	48%
(WebSphereMQ v6.0)	(1800)		1743 (1766)	0.365 (0.365)	48% (47%)

Table 5 – 1 round trip per driving application per second, client channels

Note: The large bold numbers in the table above show the WebSphere MQ V7.0 number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison with Version 6.

2.3 Distributed Queuing Test Scenario

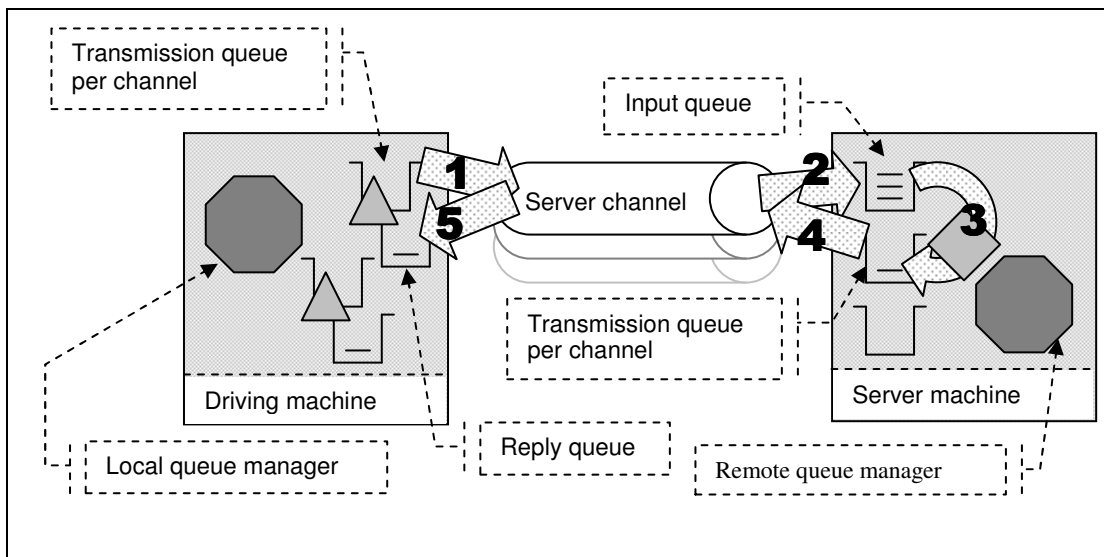


Figure 9 – Server channels between two queue managers

- 1) The Requester application puts a message to a local definition of a remote queue located on the server machine, and holds on to the message identifier returned in the message descriptor. The Requester application then waits indefinitely for a reply to arrive on a local queue.
- 2) The message channel agent takes messages off the channel and places them on the common input queue on the server machine.
- 3) The Responder application gets messages from the common input queue, and places a reply to the queue name extracted from the messages descriptor (the name of a local definition of a remote queue located on the driving machine). The queue manager copies over the message identifier from the request message to the correlation identifier of the reply message.
- 4) The message channel agent takes messages off the transmission queue and sends them over the channel to the driving machine.
- 5) The Requester application gets a reply from a local queue. The Requester application uses the message identifier held from when the request message was put to the local definition of the remote queue, as the correlation identifier in the message descriptor

Nonpersistent and persistent messages were used in the distributed queuing tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in the “*Large Messages*” section.

Application Bindings of the Responder program are ‘Shared’ , the Requester program is normally ‘Trusted’ , and the channels specified as ‘MQIBindType = FASTPATH’ except in the ‘non-trusted’ scenario where both programs use ‘shared’ bindings and the channels are specified as ‘MQIBindType = NORMAL’.

2.3.1 Nonpersistent Messages – Server Channels

Figure 10 , Figure 10a, and Figure 11 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario (see Figure 9 on the previous page), and WebSphere MQ V6.0 compared to Version 7.

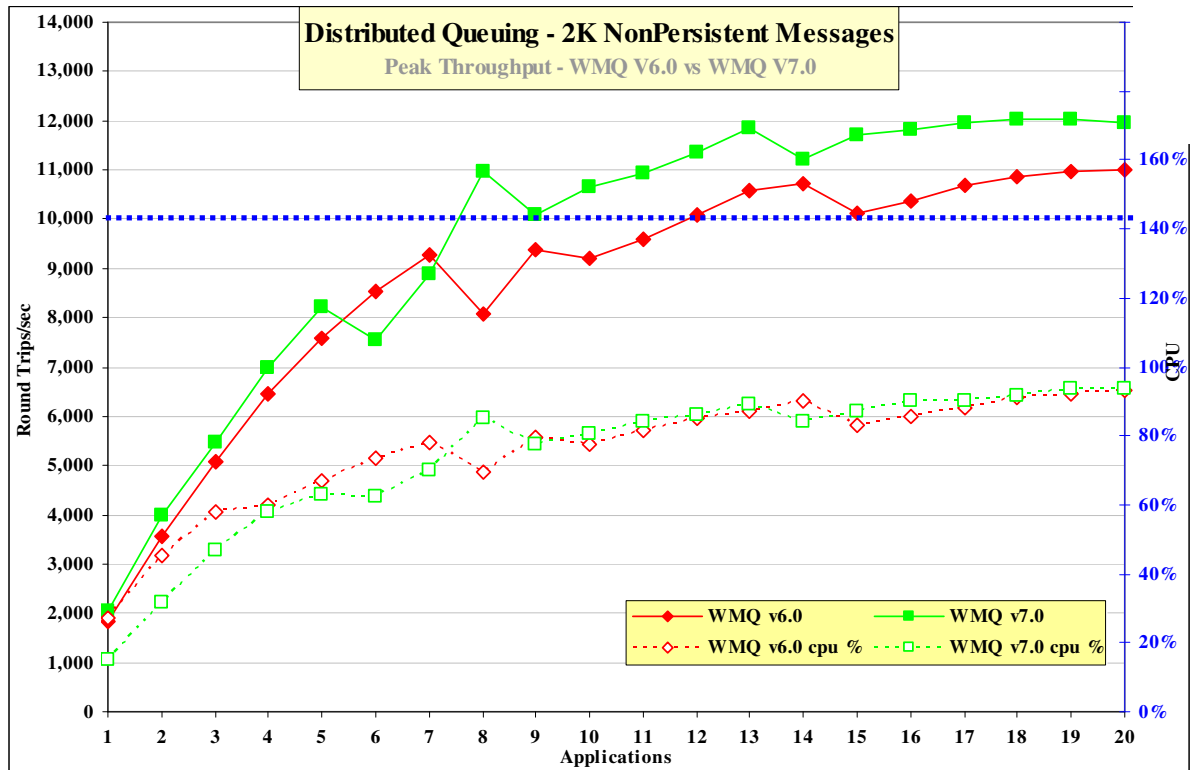


Figure 10 – Performance headline, nonpersistent messages, server channels

Note: Messaging in these tests is with no think-time.

Figure 10 and Table 6 show that the throughput of nonpersistent messages has increased by 10% when comparing Version 6.0 to Version 7.0.

Test name: dqnp	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	20	10987	0.003	93%
WebSphere MQ V7.0	18 (20)	12042 (11946)	0.002 (0.003)	92% (94%)

Table 6 – Performance headline, nonpersistent messages, server channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

2.3.2 Non Persistent non Trusted – Server Channels

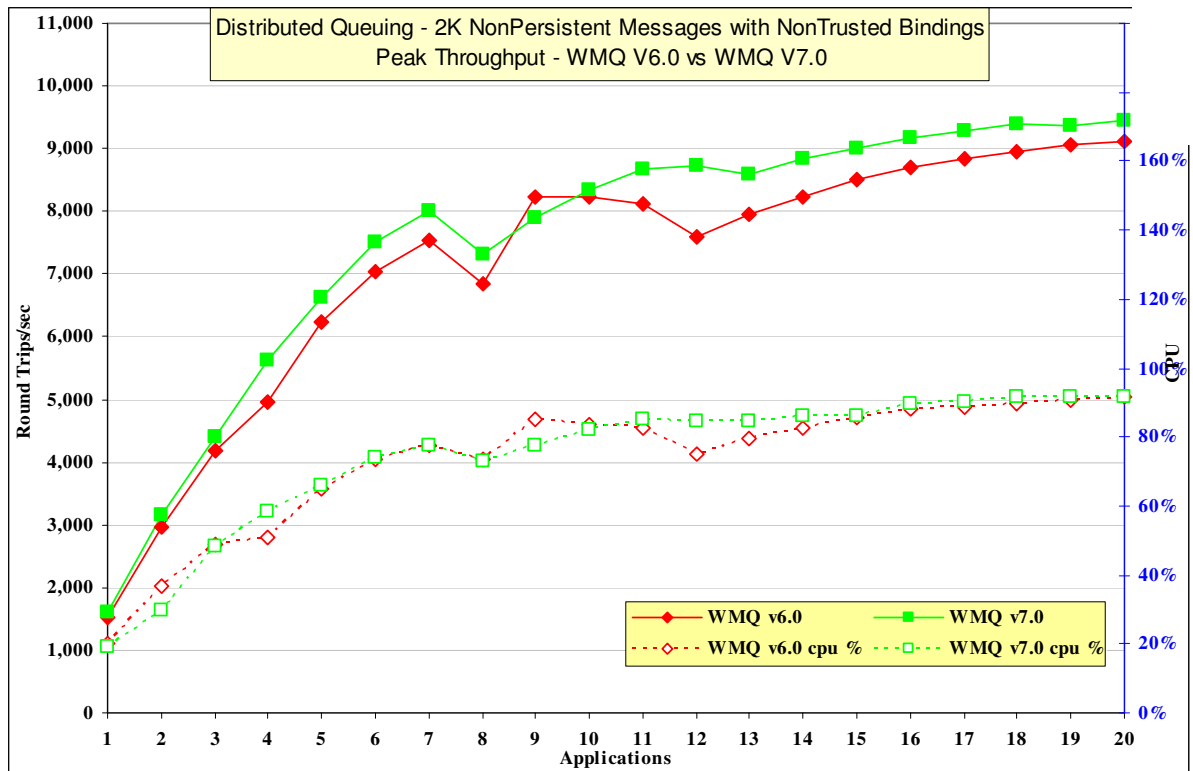


Figure 10a

Test name: Dqnp_nt	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	20	9125	0.003	92%
WebSphere MQ V7.0	20	9446	0.003	92%

The throughput of nonpersistent messages has increased by 6% when comparing Version 6.0 to Version 7.0.

2.3.3 Persistent Messages – Server Channels

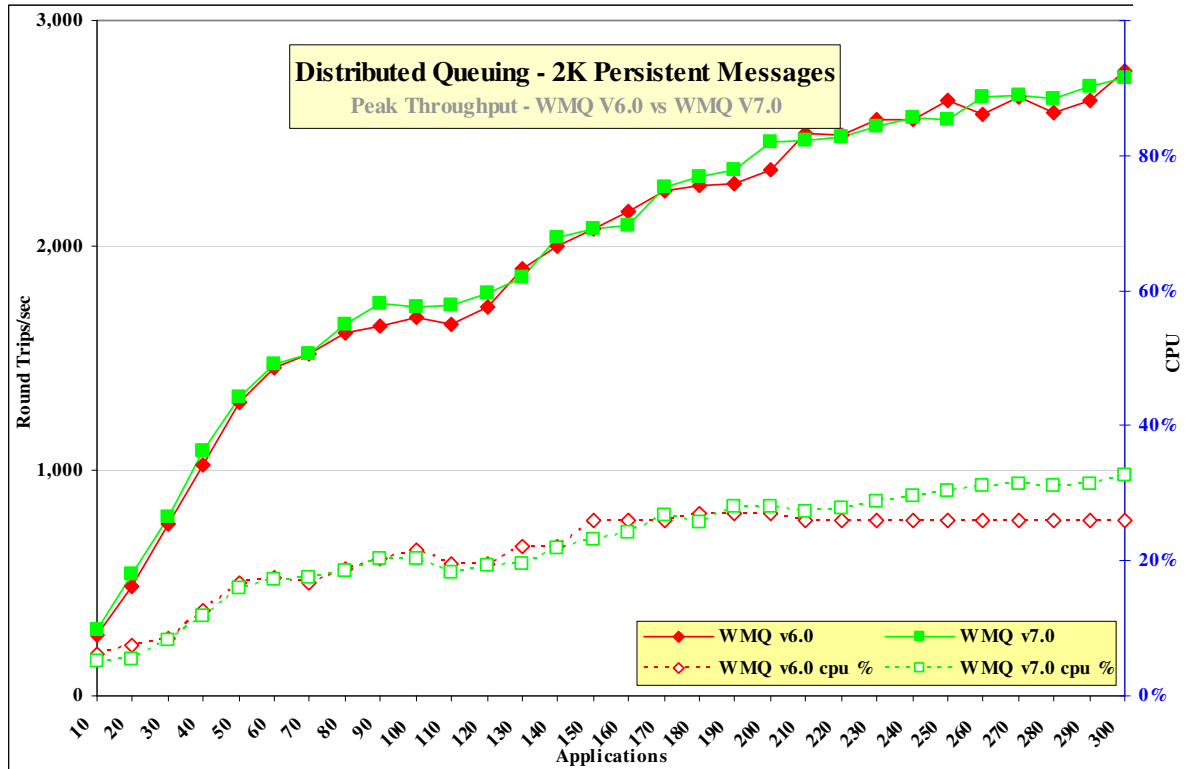


Figure 11 – Performance headline, persistent messages, server channels

Note: Messaging in these tests is with no think-time

Figure 11 and Table 7 show that the throughput of persistent messages has increased by 2% when comparing Version 6.0 to Version 7.0.

Test name: dqpm	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	300	2775	0.116	32%
WebSphere MQ V7.0	300	2747	0.128	31%

Table 7 – Performance headline, persistent messages, server channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

2.3.4 Server Channels

For the following distributed queuing measurements, the messaging rate used is 1 round trip per driving application per second, i.e. a request message outbound over the sender channel, and a reply message inbound over the receiver channel per second. Note that there are a fixed number of 4 server channel pairs for the nonpersistent messaging tests, and 2 pairs for the persistent message tests.

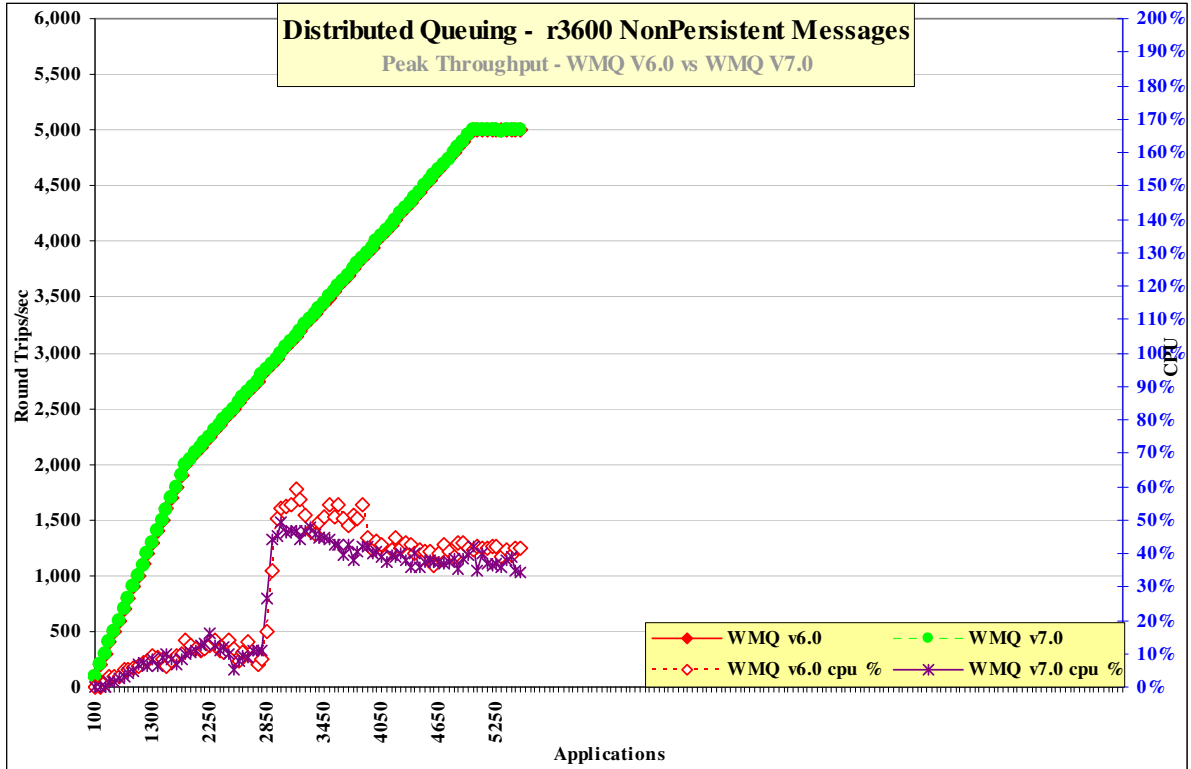


Figure 12 – 1 round trip per driving application per second, server channel, nonpersistent messages

Note: Messaging in these tests is 1 round trip per driving application per second.

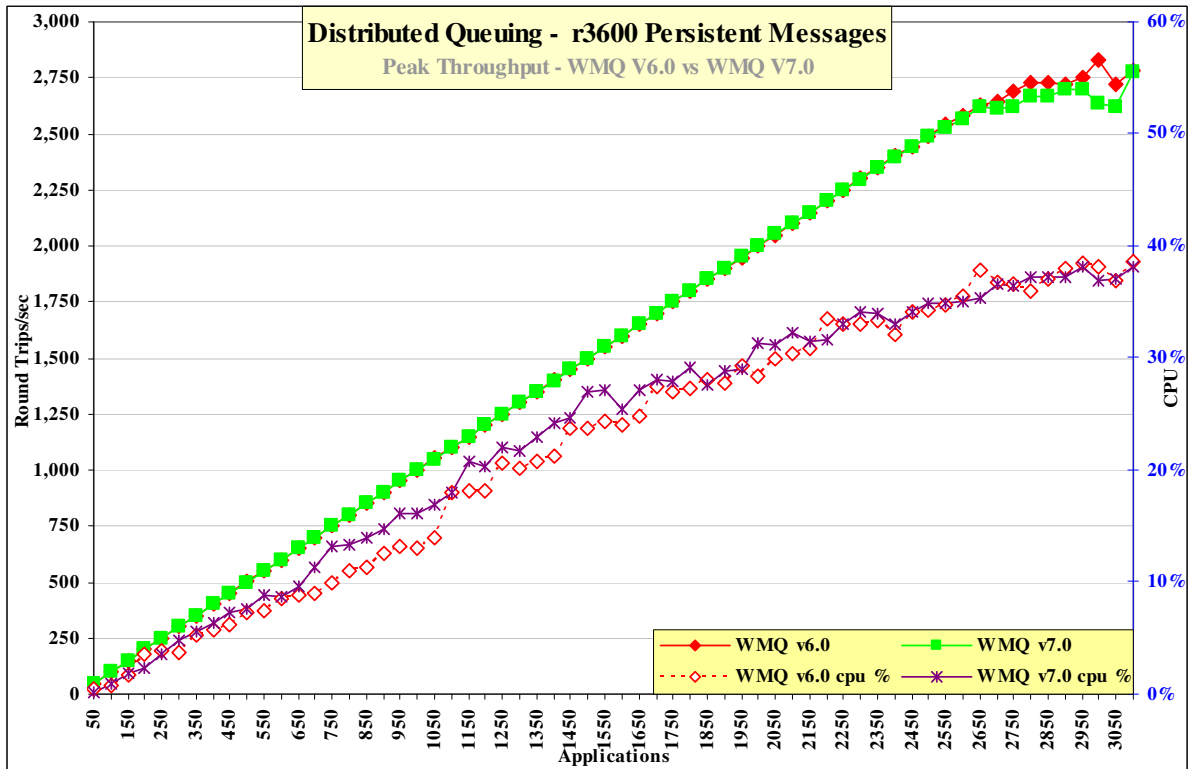


Figure 13 – 1 round trip per driving application per second, server channel, persistent messages

Figure 12, Figure 13 and Table 8 shows how WebSphere MQ V7.0 is similar in performance as version 6.0 although at a higher CPU cost.

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
dqnp_r3600 (WebSphereMQ v6.0)	5000 (5000)	3,600	4998 (4997)	0.009 (0.014)	42% (41%)
dqpm_r3600 (WebSphereMQ v6.0)	2900 (2900) (3150)	3,600	2699 (2723) (2860)	0.908 (0.863) (0.930)	37% (38%) (39%)

Table 8 – 1 round trip per driving application per second, client channels

Note: The large bold numbers in the table above show the WebSphere MQ V7.0 peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison with Version 6.

3 Large Messages

3.1 MQI Response Times: 50bytes to 100MB – Local Queue Manager

3.1.1 50bytes to 32KB

Figure 14 show that the response time for MQPUT/GET for nonpersistent message sizes between 50bytes and 32KB.

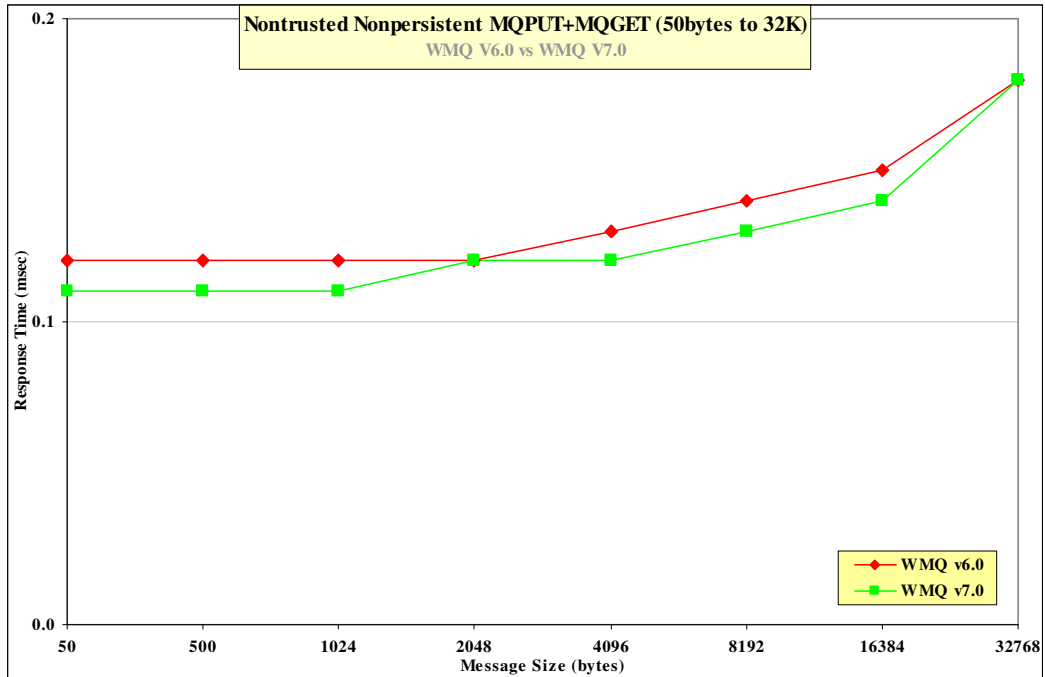
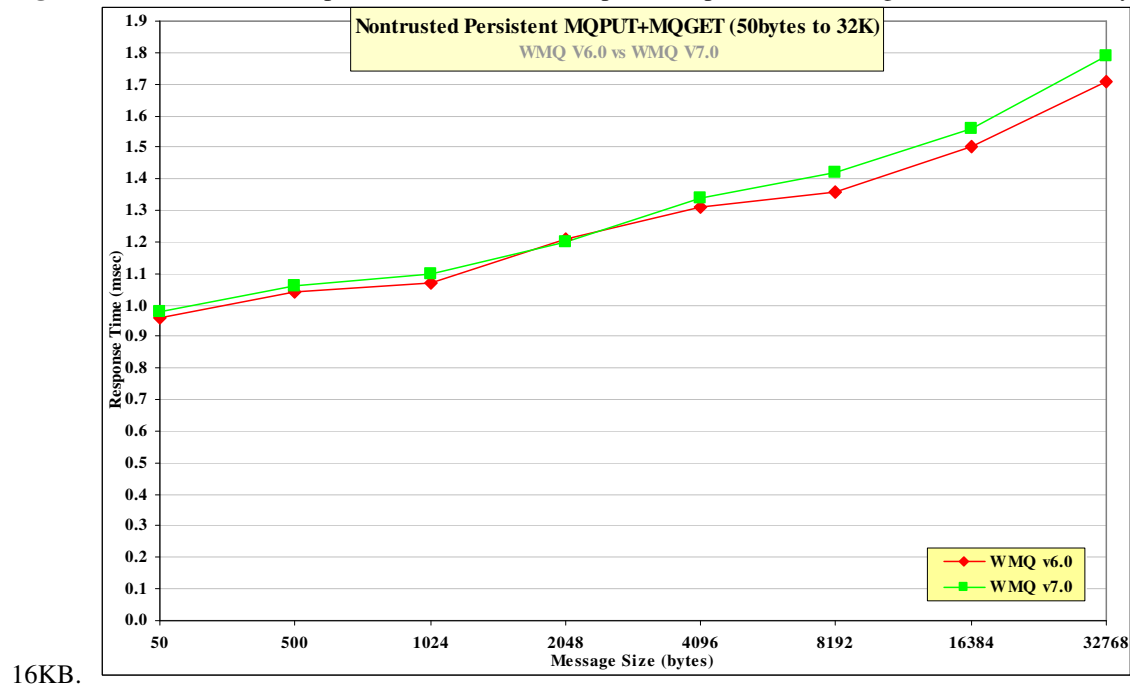


Figure 14 –The effect of nonpersistent message size on MQI response time (50byte - 32K)

Figure 15 show that the response for MQPUT/GET pairs for persistent message sizes between 50bytes and



16KB.

Figure 15 –The effect of persistent message size on MQI response time (50byte - 32K)

3.1.2 32KB to 2MB

Figure 16 show that the response time for MQPUT/GET pairs has improved for all nonpersistent message sizes between 32KB and 2MB.

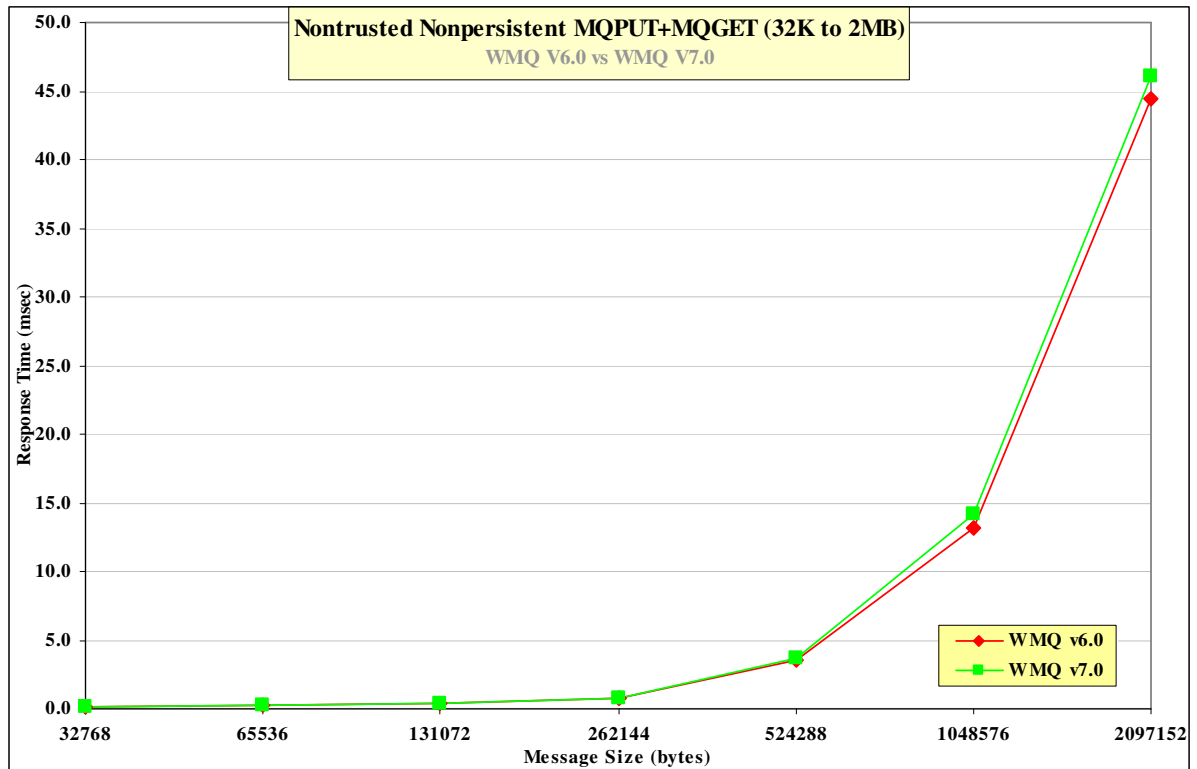


Figure 16 –The effect of nonpersistent message size on MQI response time (32K – 2MB)

Figure 17 show that the response for MQPUT/GET pairs for persistent message sizes between 32KB and 2MB.

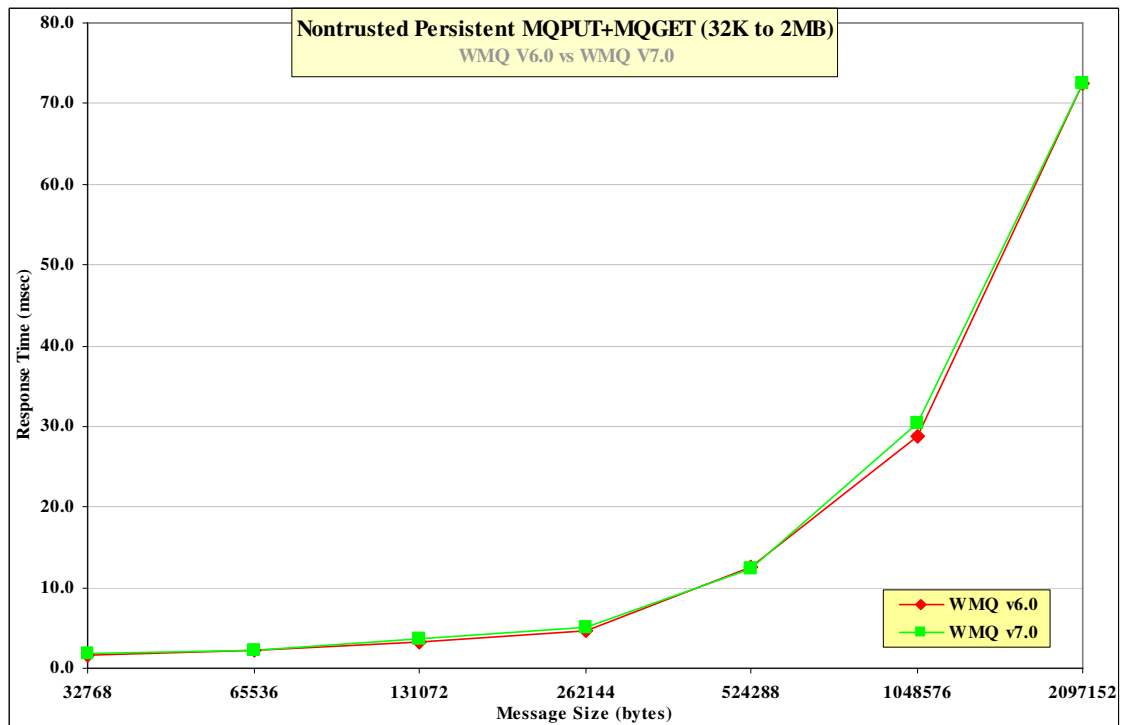


Figure 17 –The effect of persistent message size on MQI response time (32K – 2MB)

3.1.3 2MB to 100MB

Figure 18 Response time for MQPUT/GET pairs for NP message between 2MB and 100Mb.

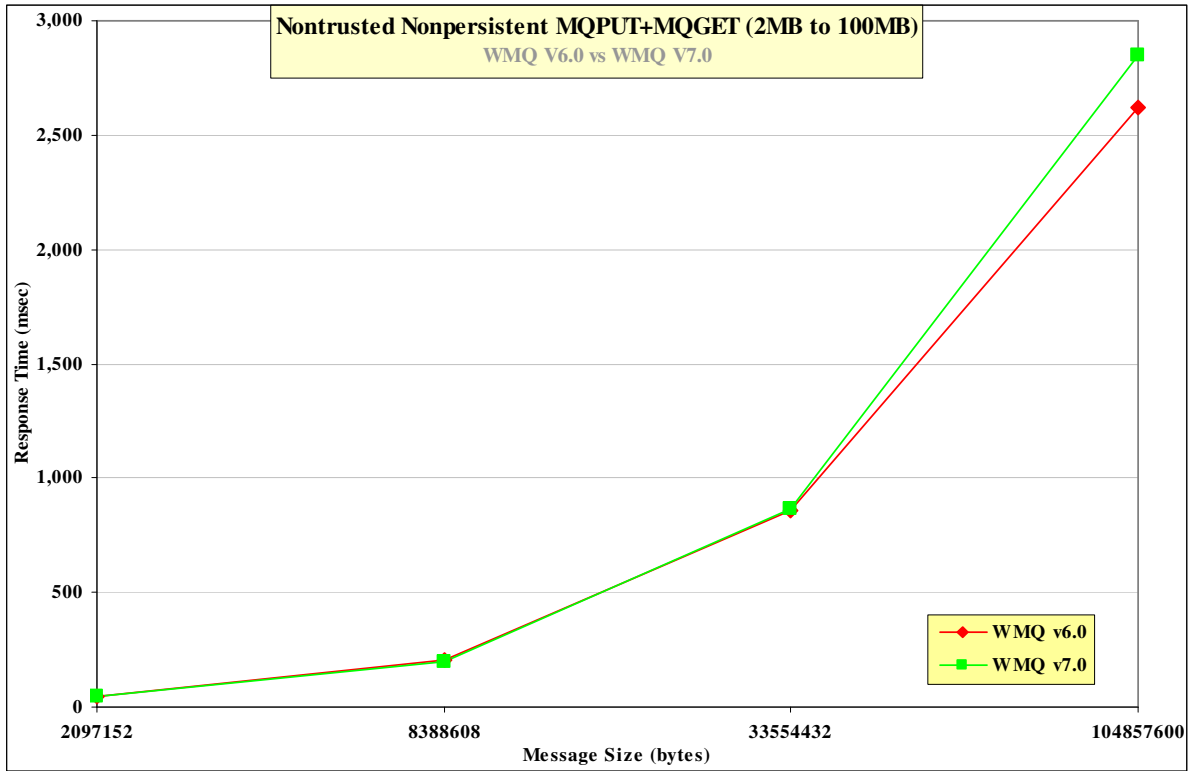


Figure 18 –The effect of nonpersistent message size on MQI response time (2MB – 100MB)

Figure 19 The response for MQPUT/GET pairs for persistent message sizes between 2MB and 100MB.

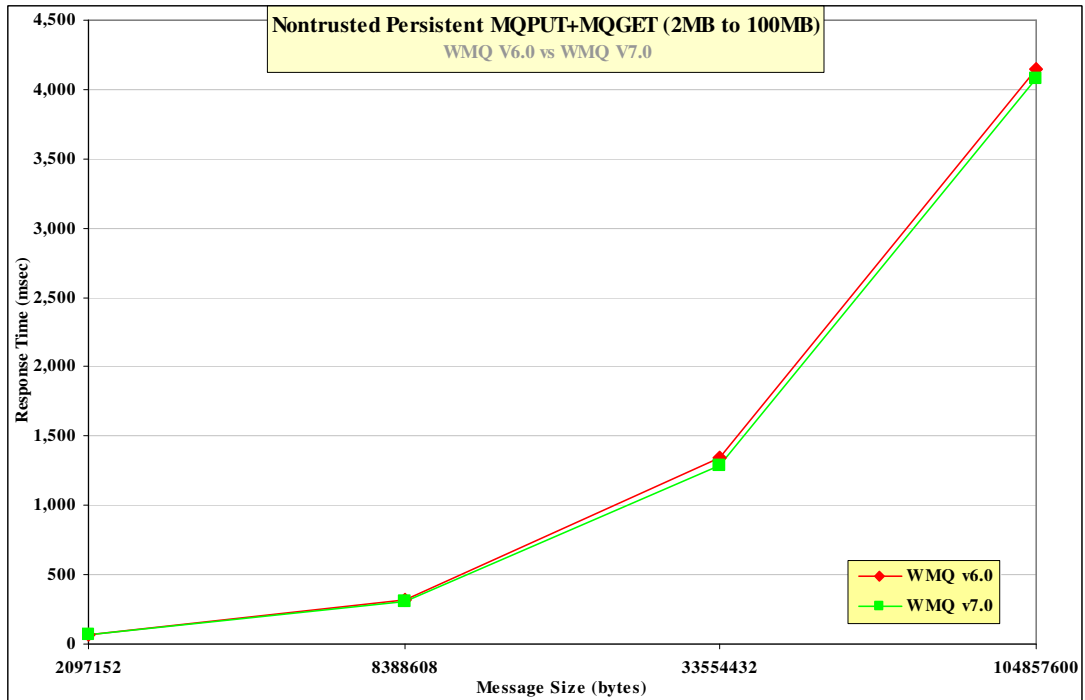


Figure 19 –The effect of persistent message size on MQI response time (2MB – 100MB)

3.2 20K Messages

3.2.1 Local Queue Manager

Figure 20 and Figure 21 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

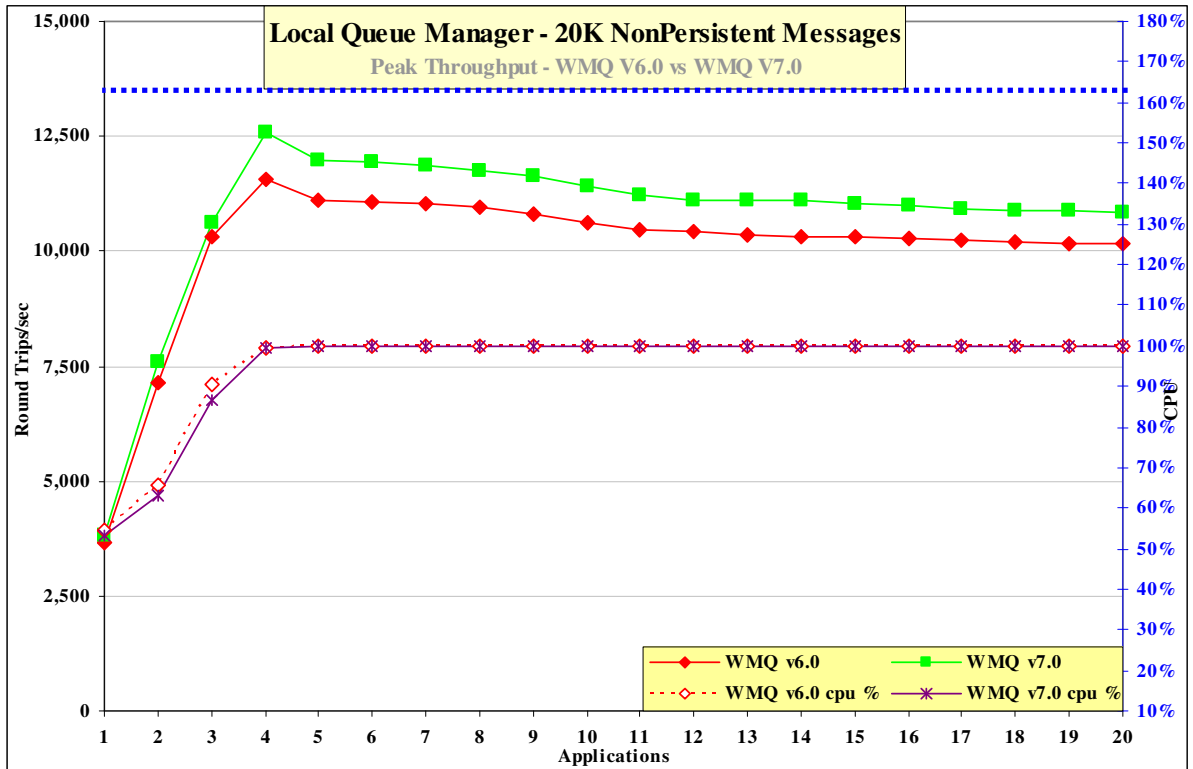


Figure 20 – 20K nonpersistent messages, local queue manager

Figure 20 and Table 9 show that the throughput of nonpersistent messages has improved by 6% comparing Version 7.0 to Version 6.0.

Test name: local_np_20K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	4	11566	0.001	100%
WebSphere MQ V7.0	4	12564	0.001	100%

Table 9 – 20K nonpersistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7

3.2.1.1 Persistent Messages

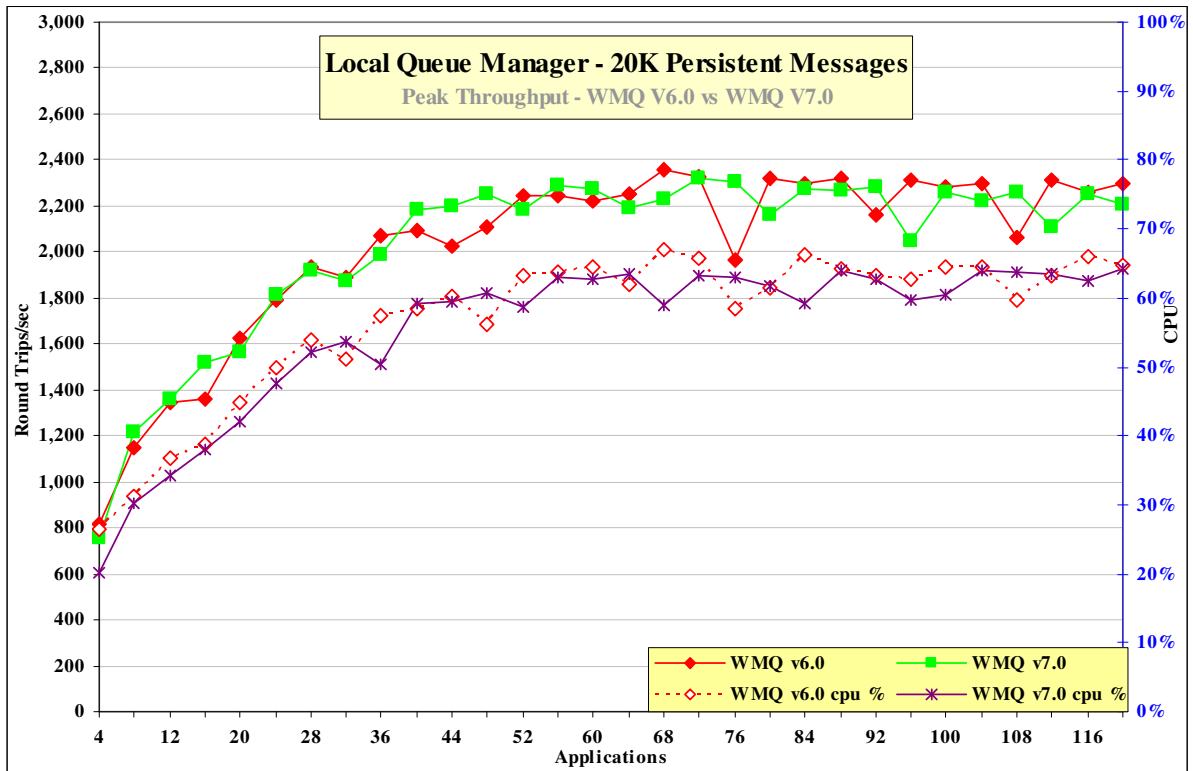


Figure 21 – 20K persistent messages, local queue manager

Figure 21 and Table 10 show that the throughput of persistent messages is similar when comparing Version 6.0 to Version 7.0.

Test name: local_pm_20K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	68	2358	0.034	67%
WebSphere MQ V7.0	(68) 72	(2227) 2321	(0.036) 0.038	(59%) 63%

Table 10 – 20K persistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.2.2 Client Channel

Figure 22 and Figure 23 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario. Clients use Optimized connections on MQ V7 in this chapter.

3.2.2.1 Nonpersistent Messages

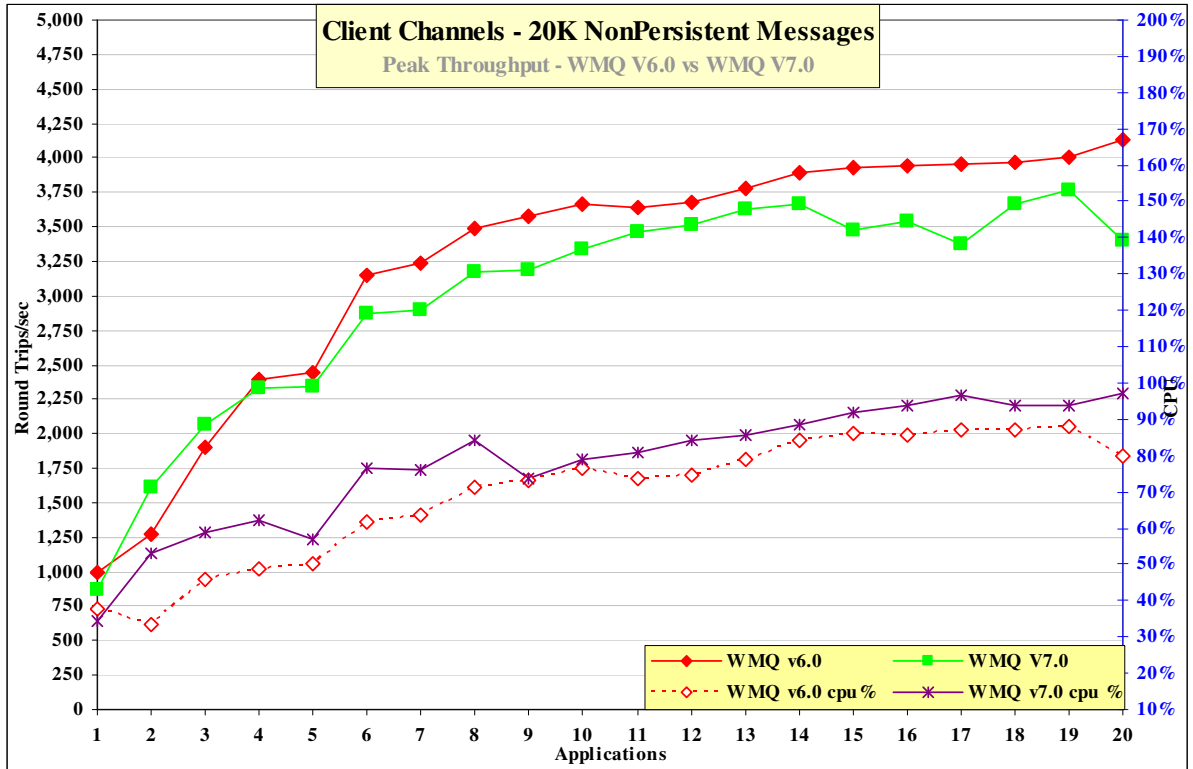


Figure 22 – 20K nonpersistent messages, client channels

Figure 22 and Table 11 show that the throughput of nonpersistent messages has degraded by 6% when comparing Version 6.0 to Version 7.0.

Test name: cInp_20K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	20	4127	0.006	88%
WebSphere MQ V7.0	19 (20)	3766 (3404)	0.006 (0.006)	94% (97%)

Table 11 – 20K nonpersistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.2.2.2 Persistent Messages

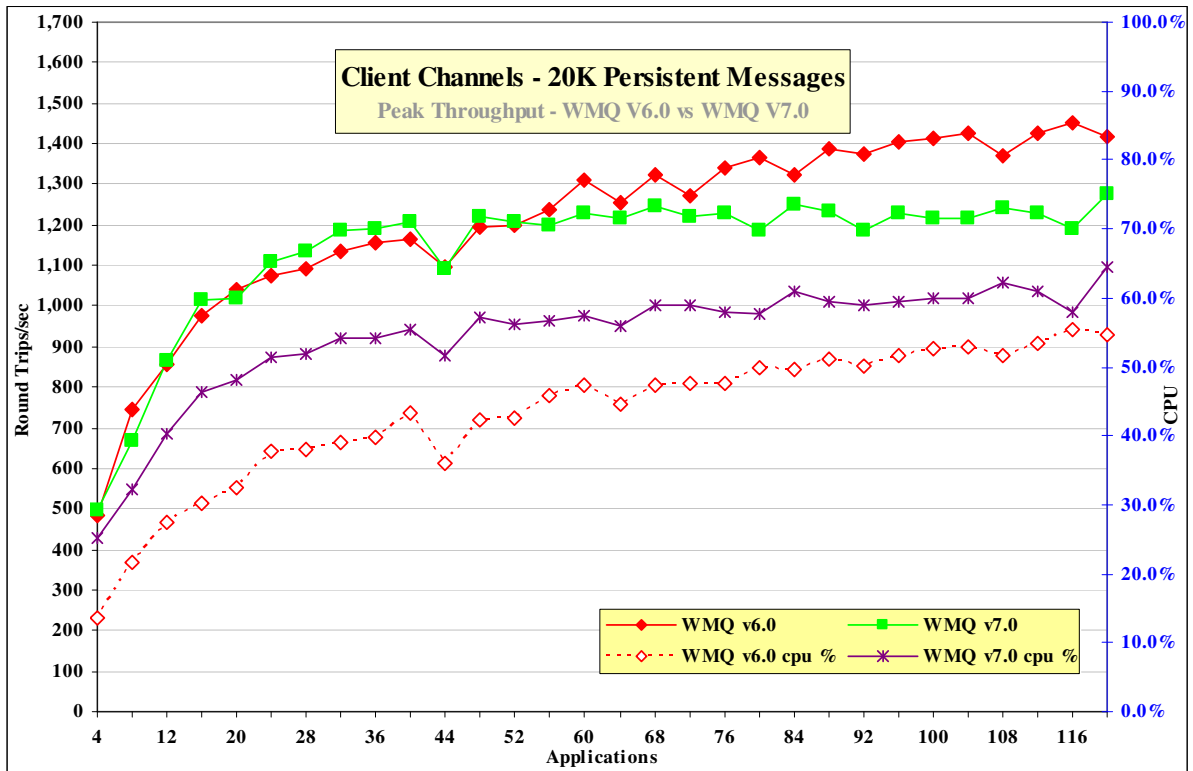


Figure 23 – 20K persistent messages, client channels

Figure 23 and Table 12 show that the throughput of persistent messages has degraded by 5% when comparing Version 6.0 to Version 7.0.

Test name: clpm_20K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	116	1450	0.083	56%
WebSphere MQ V7.0	(116) 120	(1191) 1275	(0.118) 0.113	(58%) 64%

Table 12 – 20K persistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.2.3 Distributed Queuing

Figure 24 and Figure 25 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario

3.2.3.1 Nonpersistent Messages

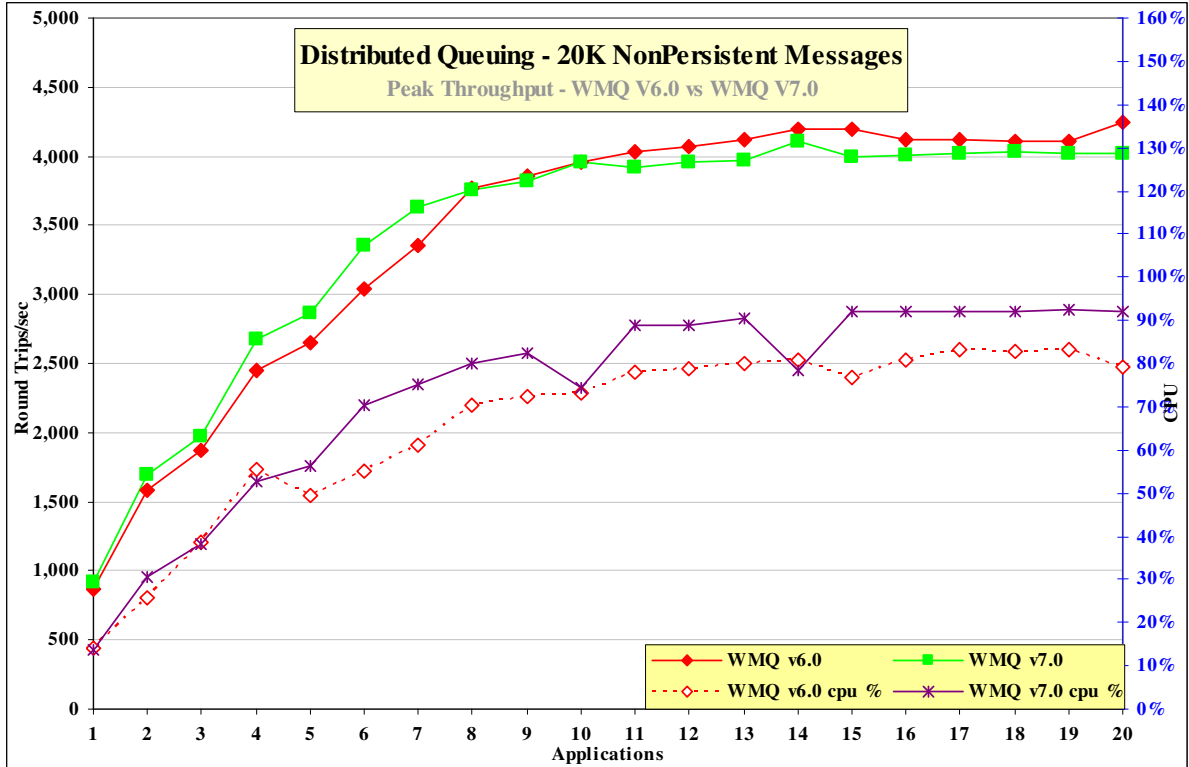


Figure 24 – 20K nonpersistent messages, distributed queuing

Figure 24 and Table 13 show that the throughput of nonpersistent messages is similar when comparing Version 6.0 to Version 7.0.

Test name: dqnp_20K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	20	13267	0.003	86%
WebSphere MQ V7.0	18 (20)	14904 (14615)	0.002 (0.002)	80% (83%)

Table 13 – 20K nonpersistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

The similar throughput between V53 and V6 which is also similar to client non persistent suggests that the bottleneck is outside the MQ layer.

3.2.3.2 Persistent Messages

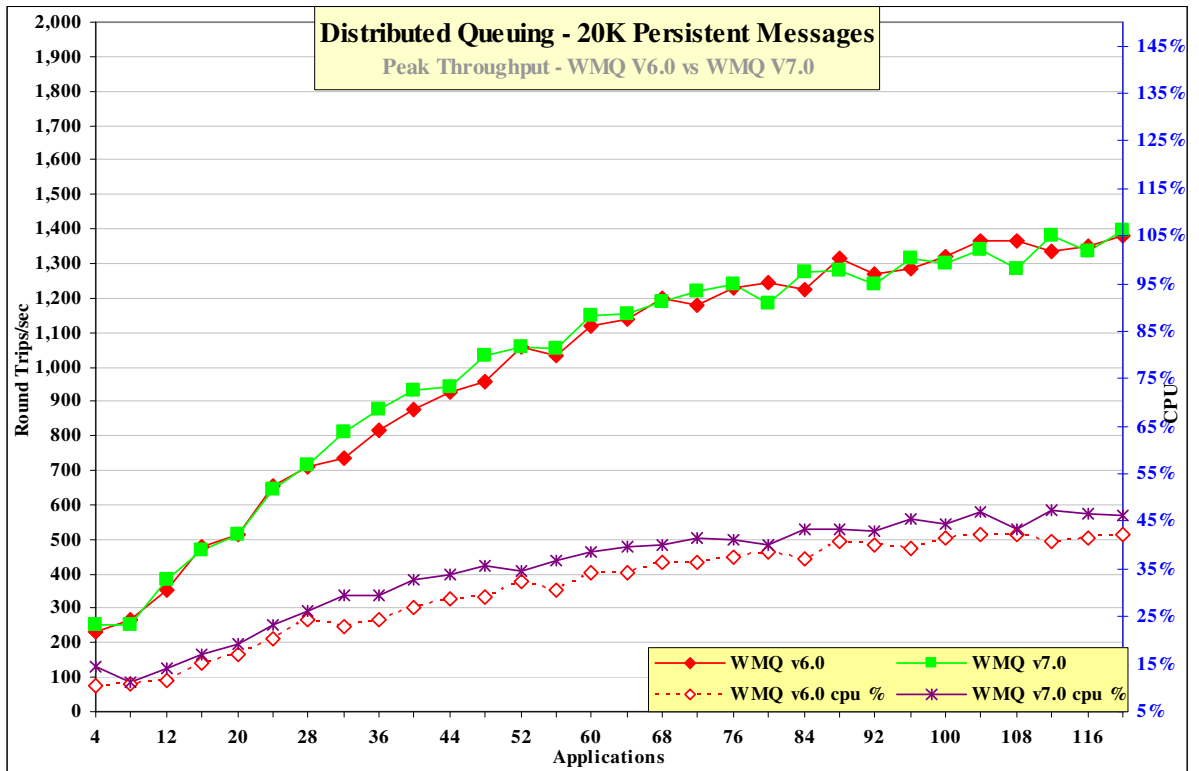


Figure 25 – 20K persistent messages, distributed queuing

Figure 25 and Table 14 show that the throughput of nonpersistent messages is similar when comparing Version 6.0 to Version 7.0.

Test name: dqpm_20K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	120	1382	0.106	42%
WebSphere MQ V7.0	120	1393	0.107	46%

Table 14 – 20K persistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.3 200K Messages

3.3.1 Local Queue Manager

Figure 26 and Figure 27 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

3.3.1.1 Nonpersistent Messages

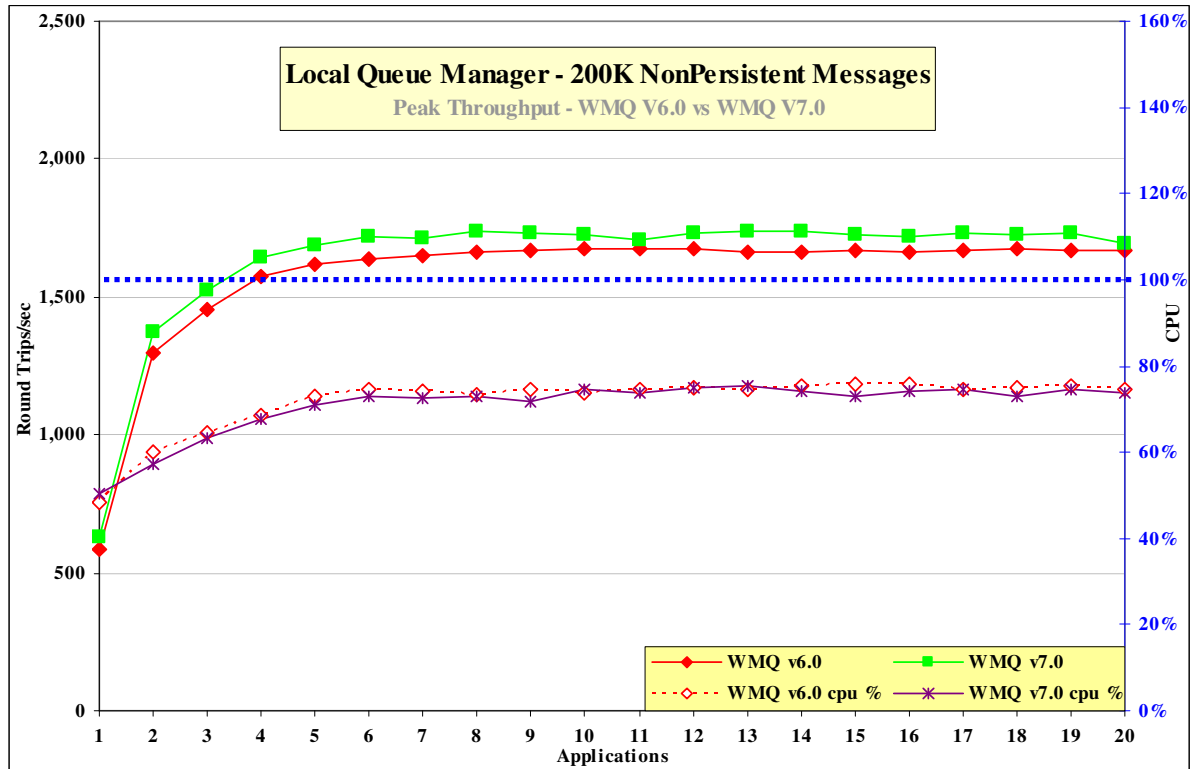


Figure 26 – 200K nonpersistent messages, local queue manager

Figure 26 and Table 15 show that the throughput of nonpersistent messages has improved by 4% when comparing Version 6.0 to Version 7.0.

Test name: local_np_200K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	11	1675	0.008	75%
WebSphere MQ V7.0	(11) 13	(1709) 1739	(0.008) 0.008	(74%) 75%

Table 15 – 200K nonpersistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7

3.3.1.2 Persistent Messages

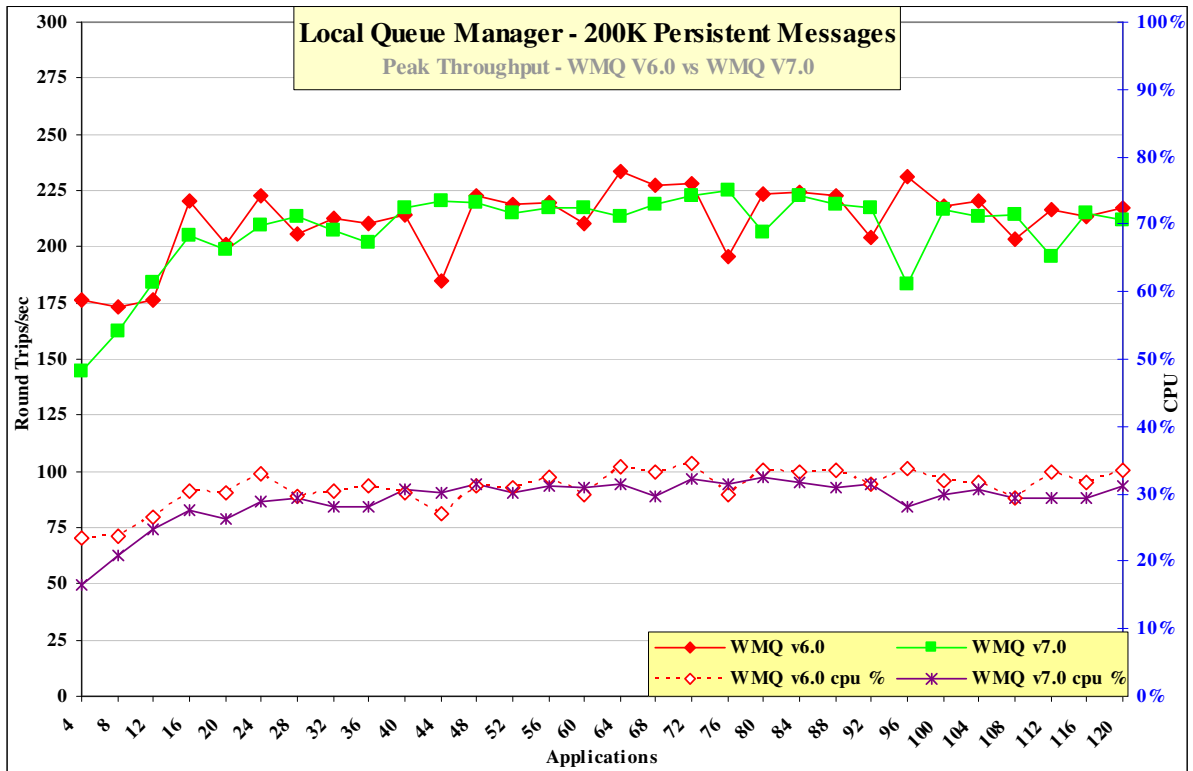


Figure 27 – 200K persistent messages, local queue manager

Figure 27 and Table 16 show that the throughput of persistent messages is similar comparing Version 6.0 to Version 7.0.

Test name: local_pm_200K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	64	234	0.329	34%
WebSphere MQ V7.0	(64) 76	(213) 225	(0.337) 0.389	(31%) 31%

Table 16 – 200K persistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.3.2 Client Channel

Figure 28 and Figure 29 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario. Clients in Chapter 3 used Optimized connections on MQ V7.

3.3.2.1 Nonpersistent Messages

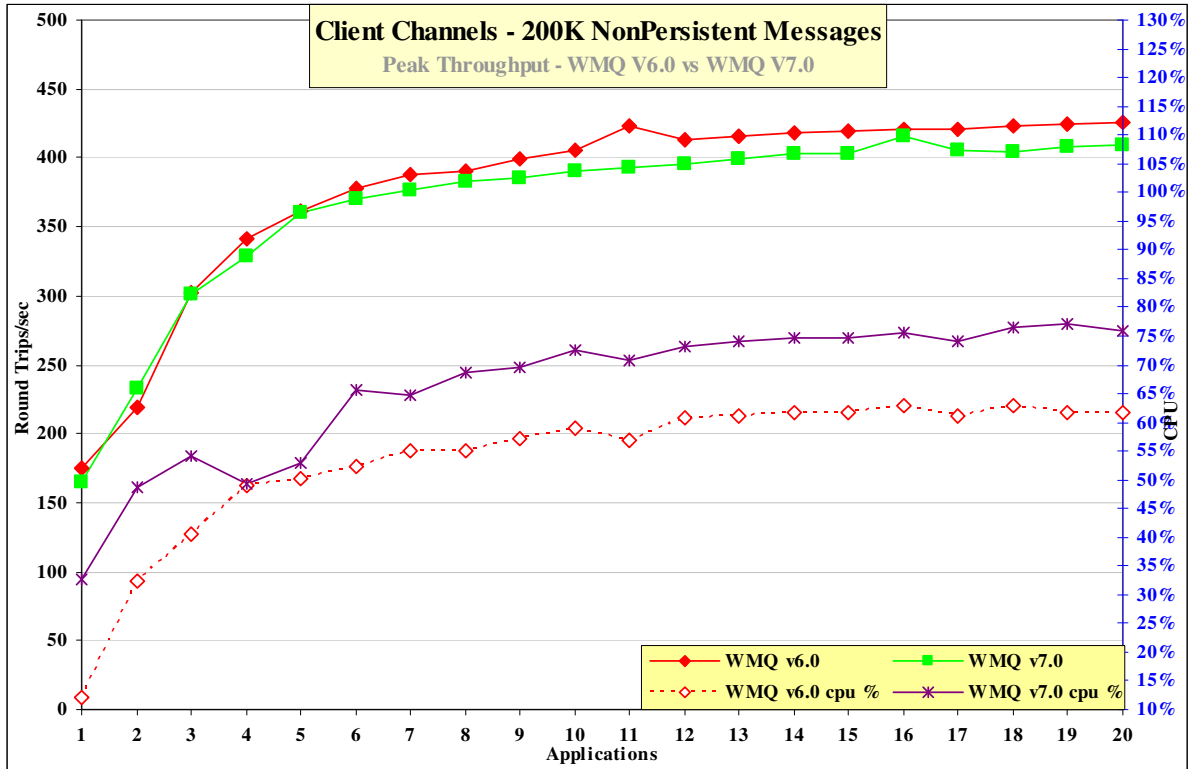


Figure 28 – 200K nonpersistent messages, client channels

Figure 28 and Table 17 show that the throughput of nonpersistent messages has degraded by 2 % when comparing Version 6.0 to Version 7.0 but uses more CPU.

Test name: clnp_200K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	20	426	0.054	62%
WebSphere MQ V7.0	16 (20)	416 (409)	0.045 (0.059)	76% (76%)

Table 17 – 200K nonpersistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput.

3.3.2.2 Persistent Messages

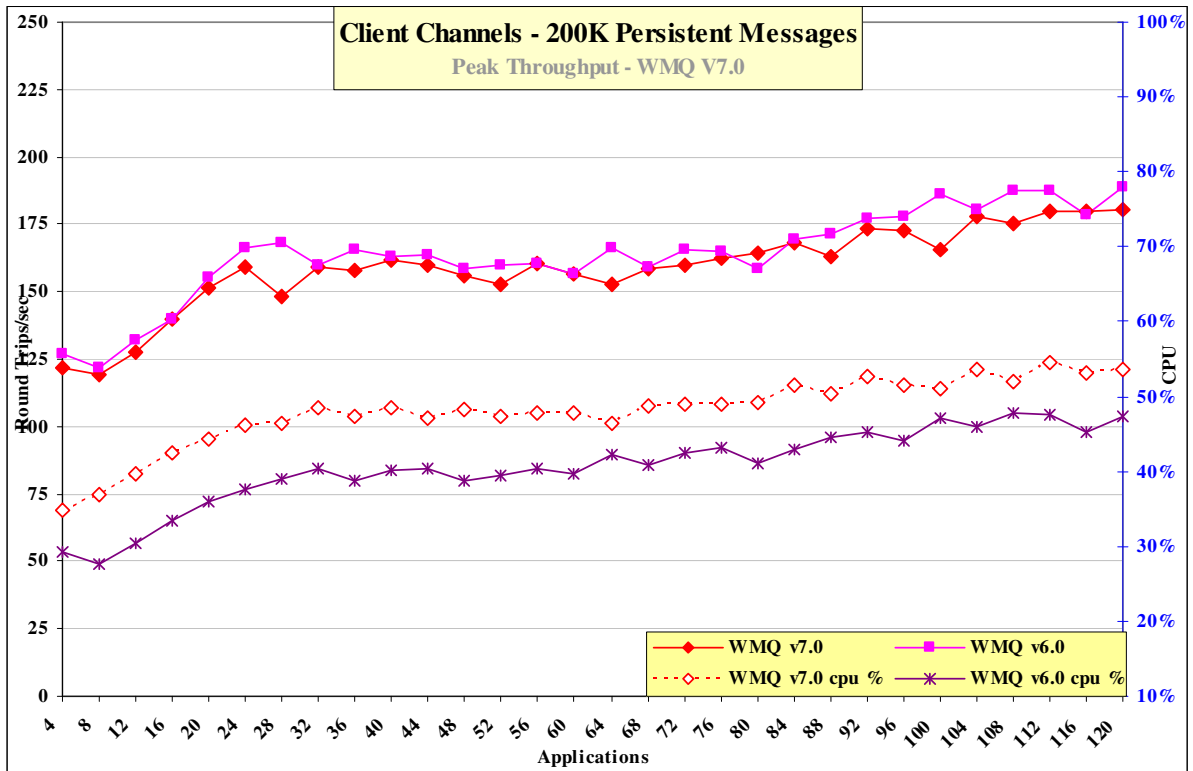


Figure 29 – 200K persistent messages, client channels

Figure 29 and Table 18 only show the throughput of persistent messaging is similar when comparing Version 6 with V7.0 message rates

Test name: clpm_200K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	120	189	0.743	48%
WebSphere MQ V7.0	120	181	0.795	54%

Table 18 – 200K persistent messages, client channels

3.3.3 Distributed Queuing

Figure 30 and Figure 31 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario

3.3.3.1 Nonpersistent Messages

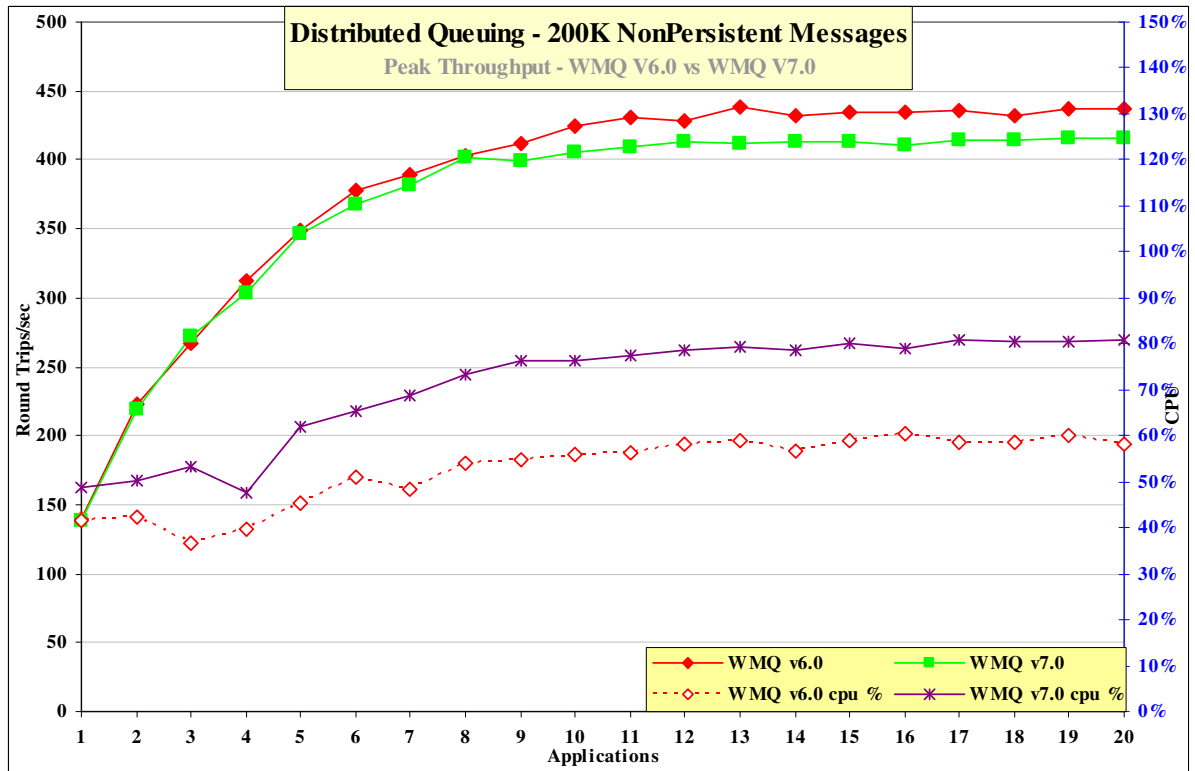


Figure 30 – 200K nonpersistent messages, distributed queuing

Figure 30 and Table 19 show that the throughput of nonpersistent messages has degraded by 3% when comparing Version 6.0 to Version 7.0 but uses more cpu

Test name: dqnp_200K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	13	438	0.035	59%
WebSphere MQ V7.0	(13) 19	(411) 415	(0.037) 0.055	(79%) 80%

Table 19 – 200K nonpersistent messages, distributed queuing

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput.

3.3.3.2 Persistent Messages

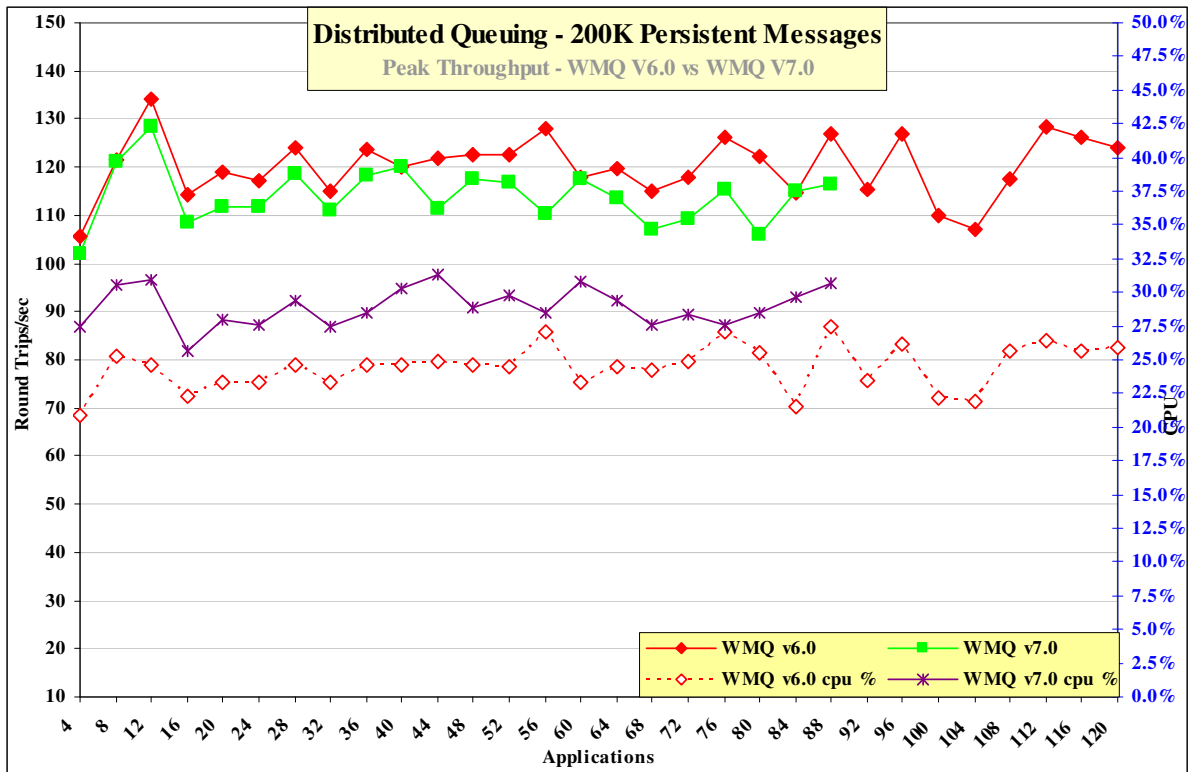


Figure 31 – 200K persistent messages, distributed queuing

Figure 31 and Table 20 show that the throughput of nonpersistent messages has degraded by 5% when comparing Version 6.0 to Version 7.0 but uses more cpu.

Test name: dqpm_200K	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	12	134	0.104	25%
WebSphere MQ V7.0	12	128	0.102	31%

Table 20 – 200K persistent messages, distributed queuing

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.4 2MB Messages

3.4.1 Local Queue Manager

Figure 32 and Figure 33 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

3.4.1.1 Nonpersistent Messages

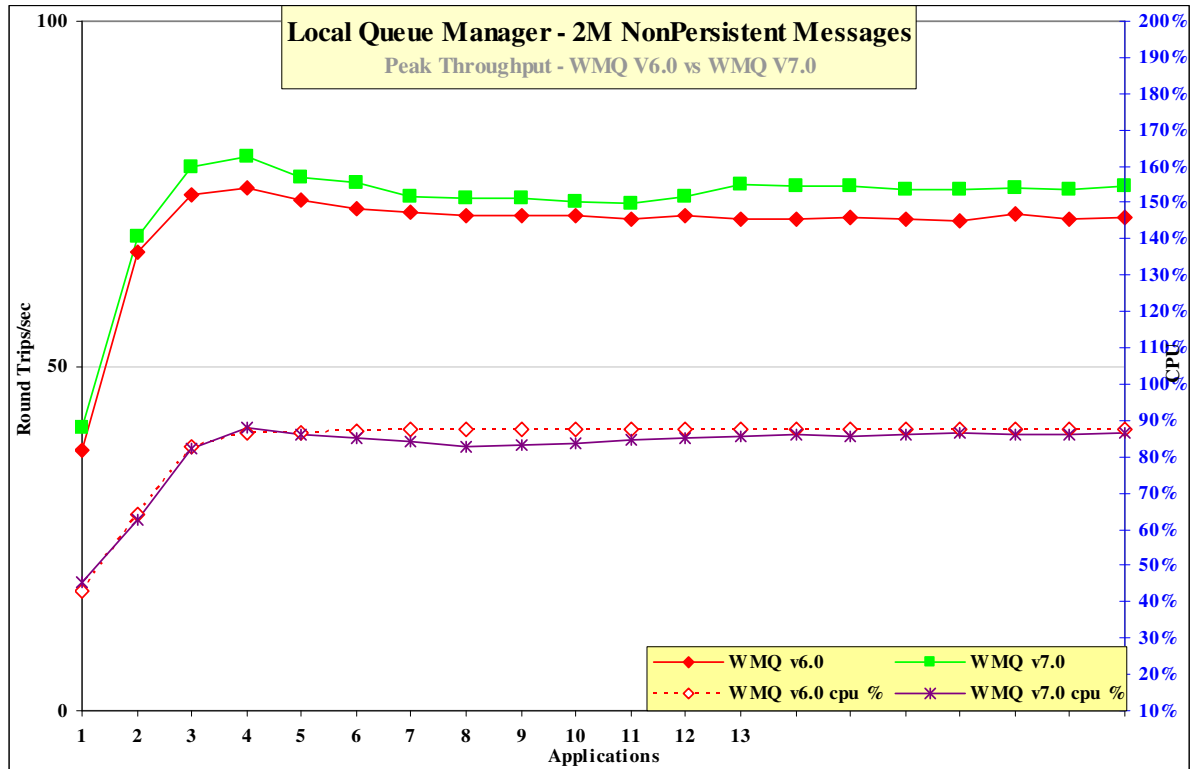


Figure 32 – 2M nonpersistent messages, local queue manager

Figure 32 and Table 21 show that the throughput of nonpersistent messages has improved by 6% when comparing Version 6.0 to Version 7.0.

Test name: local_np_2M	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	4	76	0.060	87%
WebSphere MQ V7.0	4	80	0.057	88%

Table 21 – 2M nonpersistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7

3.4.1.2 Persistent Messages

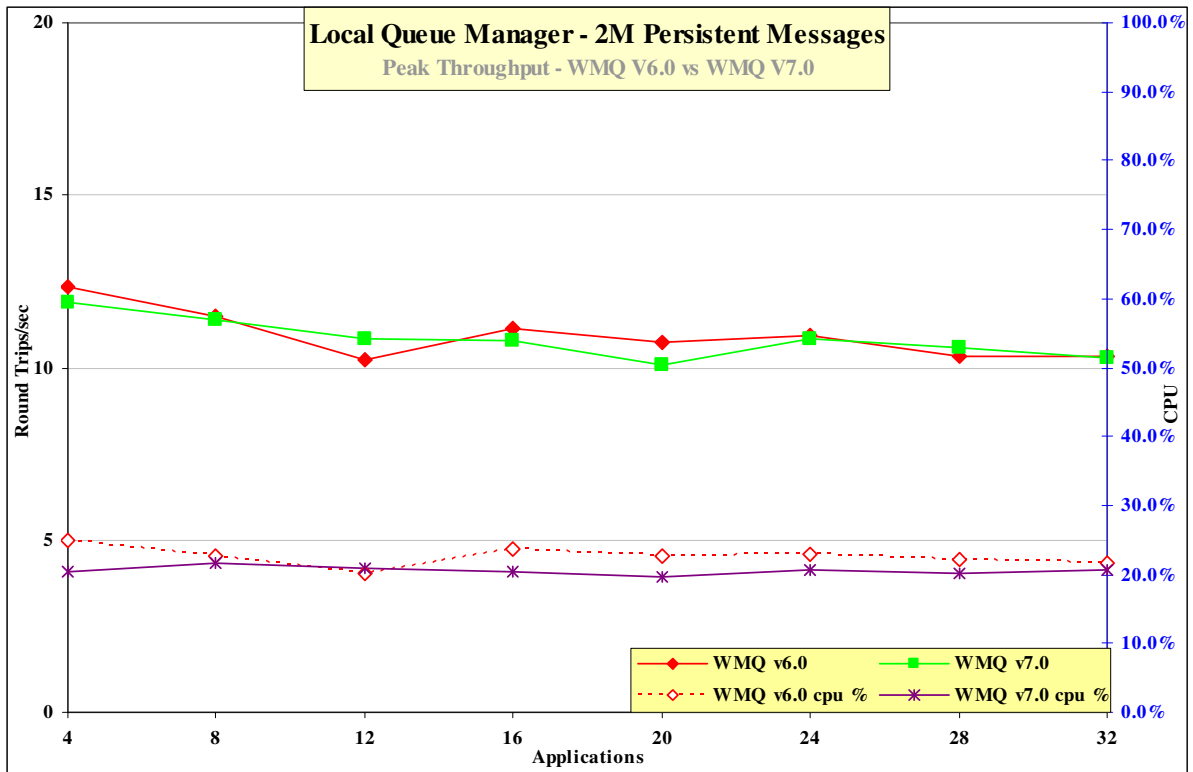


Figure 33 – 2M persistent messages, local queue manager

Figure 33 and Table 22 show that the throughput of persistent messages is similar when comparing Version 6.0 to Version 7.0.

Test name: local_pm_2M	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	4	12	0.322	25%
WebSphere MQ V7.0	4	12	0.338	20%

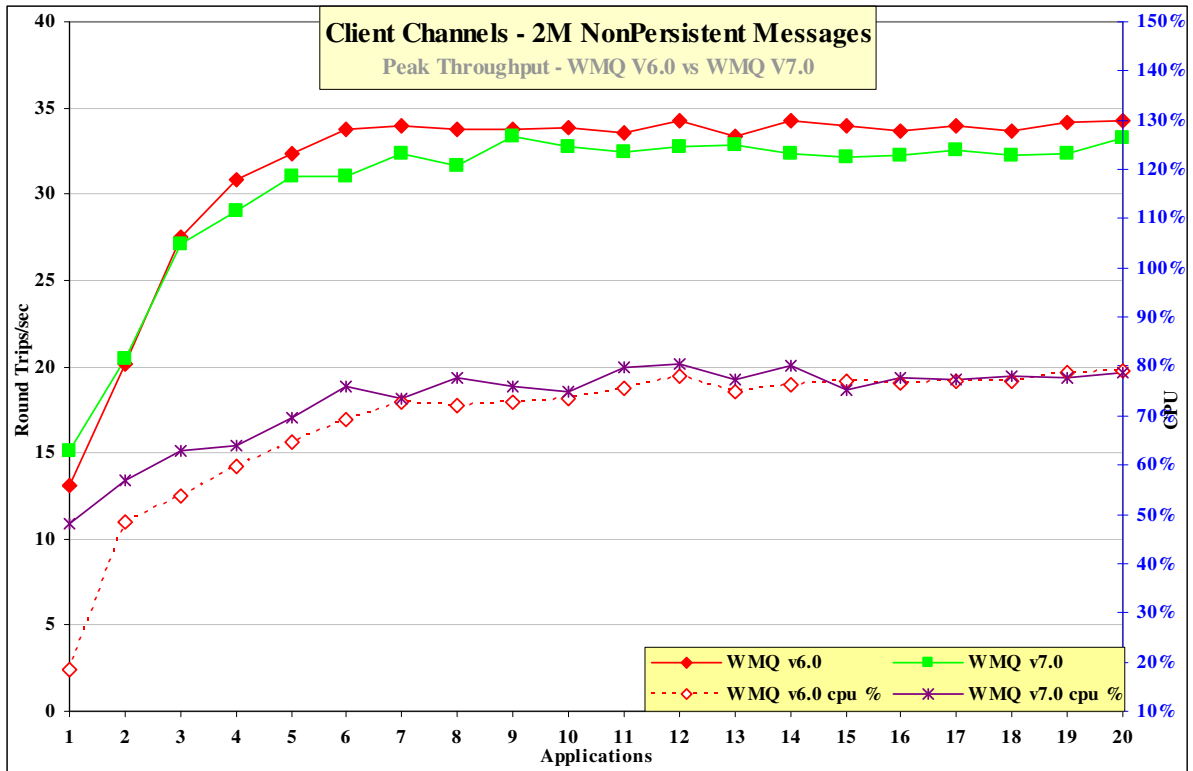
Table 22 – 2M persistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version V7.

3.4.2 Client Channel

Figure 34 and Figure 35 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario. Clients in Chapter 3 used Optimized connections on MQ V7.

3.4.2.1 Nonpersistent Messages



3.4.2.2

Figure 34 – 2M nonpersistent messages, client channels

Figure 34 and Table 23 show that the throughput of nonpersistent messages has degraded by 3% when comparing Version 6.0 to Version 7.0 but uses more cpu.

Test name: cInp_2M	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	14	34	0.491	76%
WebSphere MQ V7.0	9 (14)	33 (32)	0.318 (0.515)	76% (80%)

Table 23 – 2M nonpersistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.4.2.3 Persistent Messages

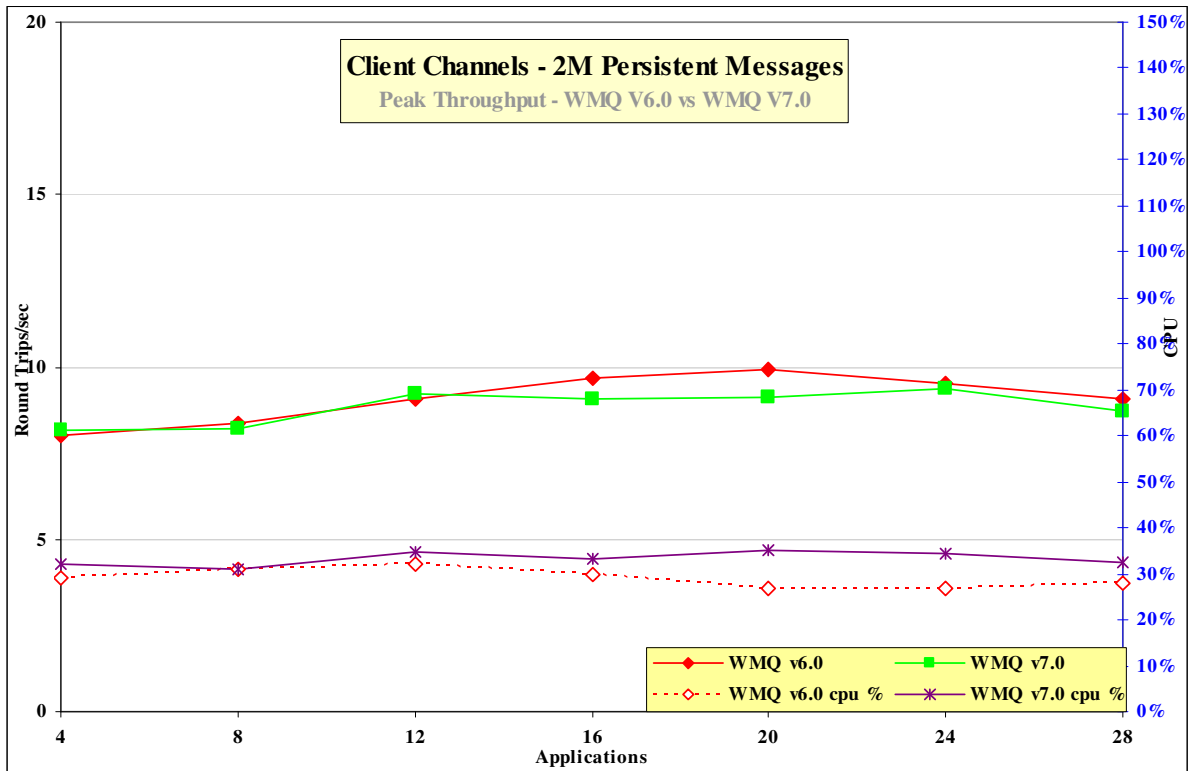


Figure 35 – 2M persistent messages, client channels

Figure 35 and Table 24 show that the throughput of persistent messages degraded by 2% when comparing Version 6.0 to Version 7.0 but uses more cpu.

Test name: clpm_2M	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	4 (20)	8 (10)	0.499 (2.34)	29% (27%)
WebSphere MQ V7.0	4 (20)	8 (9)	0.491 (2.48)	32% (35%)

Table 24 – 2M persistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.4.3 Distributed Queuing

Figure 36 and Figure 37 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario.

3.4.3.1 Nonpersistent Messages

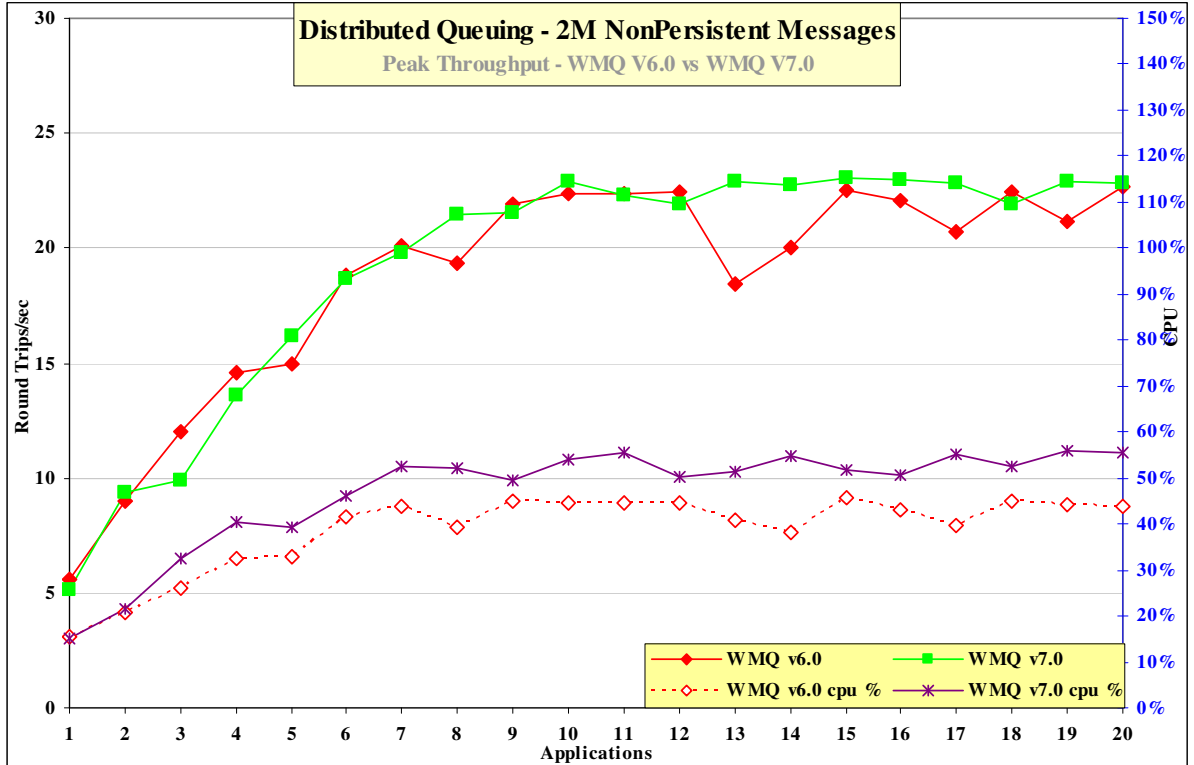


Figure 36 – 2M nonpersistent messages, distributed queuing

Figure 36 and Table 25 show that the throughput of nonpersistent messages has improved by 2% when comparing Version 6.0 to Version 7.0 although the CPU cost per message has increased.

Test name: dqnp_2M	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	12	22	0.629	46%
WebSphere MQ V7.0	12	22	0.627	52%

Table 25 – 2M nonpersistent messages, distributed queuing

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

3.4.3.2 Persistent Messages

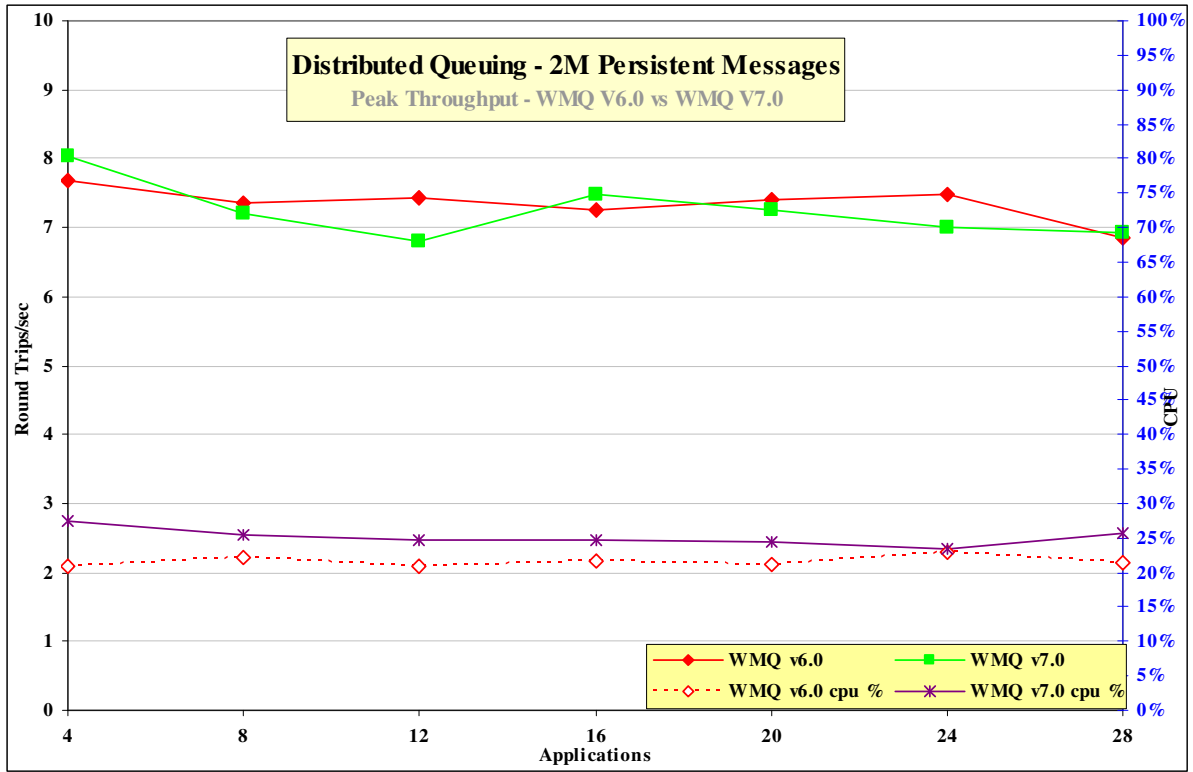


Figure 37 – 2M persistent messages, distributed queuing

Figure 37 and Table 26 show that the throughput of nonpersistent messages is similar when comparing Version 6.0 to Version 7.0.

Test name: dqpm_2M	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	4	8	0.532	21%
WebSphere MQ V7.0	4	8	0.485	27%

Table 26 – 2M persistent messages, distributed queuing

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

4 Application Bindings

This report analyzes the rate that messages can be exchanged between a Requester (Driver) application and a Responder (Server) application. This chapter looks at the effect of various combinations of application bindings for Requester and Responder programs.

	Requester	Responder
Normal	Trusted	Non Trusted
Isolated	Isolated	Isolated
Trusted	Trusted	Trusted
Non Trusted	Shared	Shared

4.1 Local Queue Manager

Figure 38 and Figure 39 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

4.1.1 Nonpersistent Messages

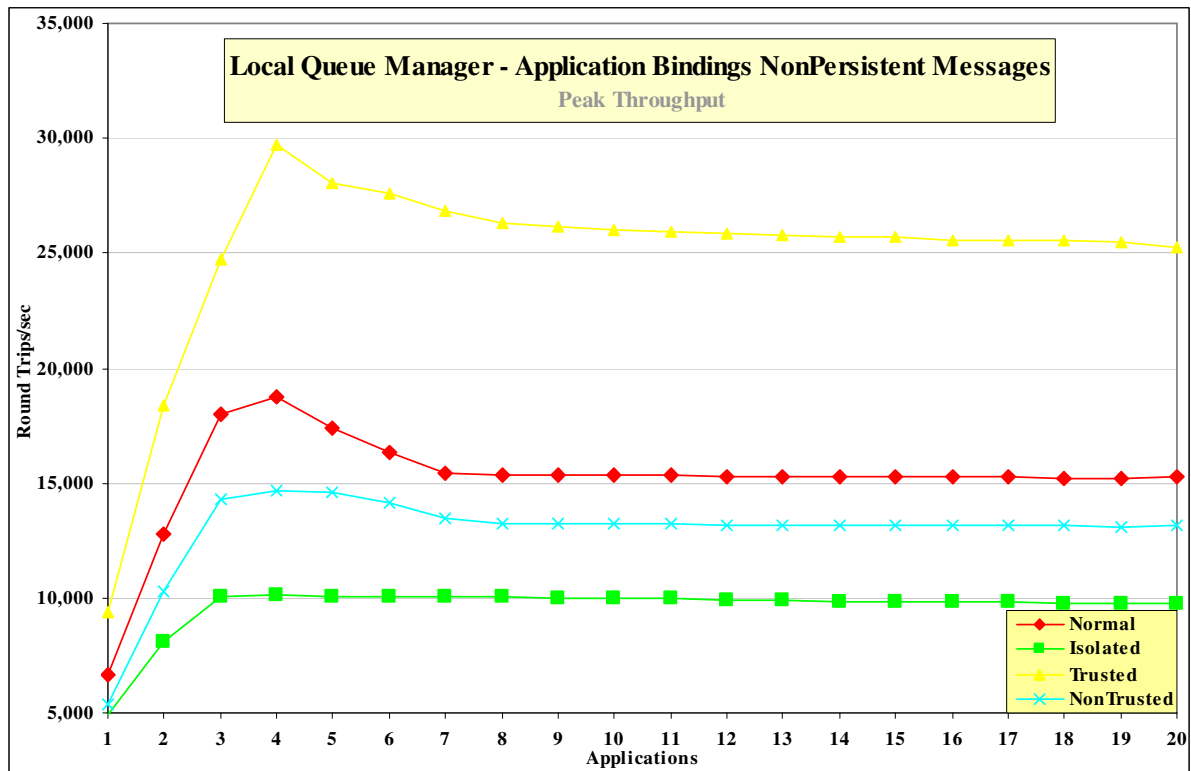


Figure 38 – Application binding, nonpersistent messages, local queue manager

Figure 38 and Table 27 show that the peak throughput of nonpersistent messages when comparing Normal, Isolated, Trusted and Shared bindings.

Test	Apps	Round Trips/sec	Response time (s)	CPU
Normal	4	18754	0.0002	100%
Isolated	4	10121	0.0005	100%
Trusted	4	29724	0.0002	98%
Shared	4	14680	0.0003	100%

Table 27 – Application binding, nonpersistent messages, local queue manager

4.1.2 Persistent Messages

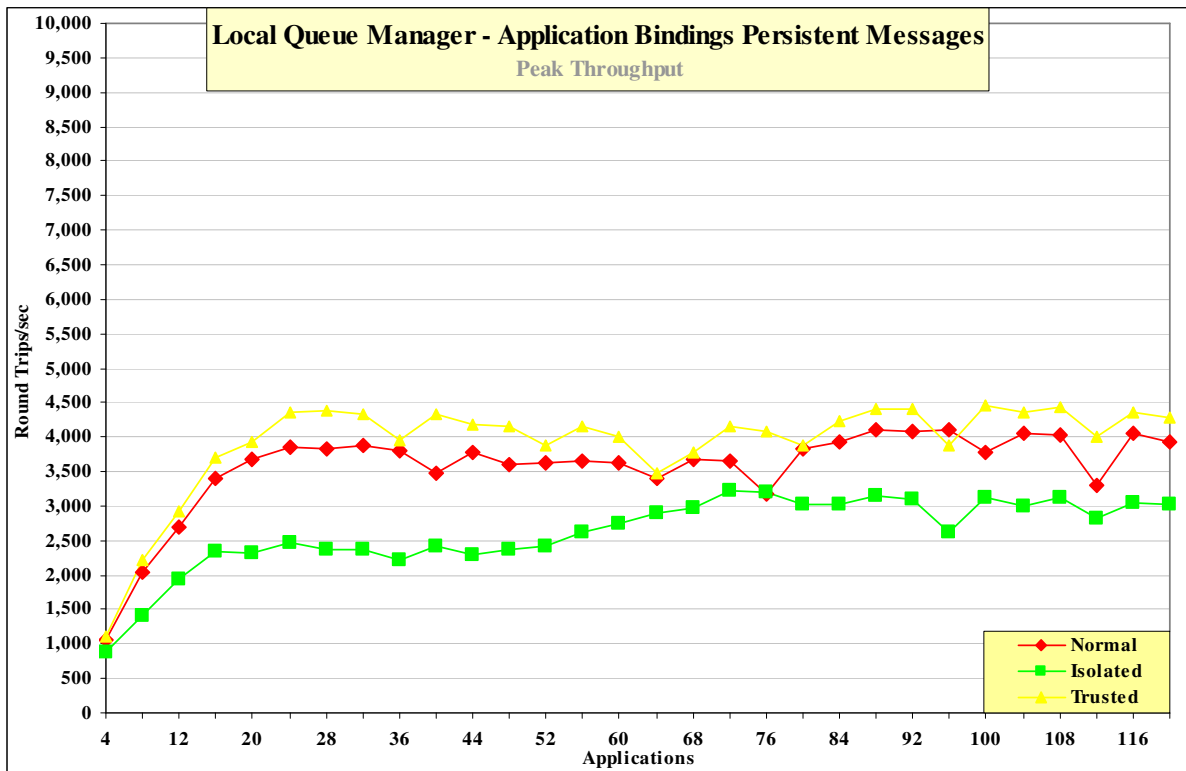


Figure 39 – Application binding, persistent messages, local queue manager

Figure 39 and Table 28 show that the peak throughput of persistent messages when comparing Normal, Isolated and Trusted bindings.

Test	Apps	Round Trips/sec	Response time (s)	CPU
Normal	(72) 88	(3652) 4106	(0.023) 0.026	(62%) 70%
Isolated	72	3220	0.028	82%
Trusted	(72) 100	(4165) 4270	(0.021) 0.035	(63%) 71%

Table 28 – Application binding, persistent messages, local queue manager

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

The difference in throughput due to using various application bindings is obscured by the I/O necessary for Persistent messages however the smaller CPU requirement needed by Trusted bindings can still be observed.

4.2 Client Channels

Figure 40 and Figure 41 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario. Clients in Chapter 4 used Optimized connections on MQ V7.

4.2.1 Nonpersistent Messages

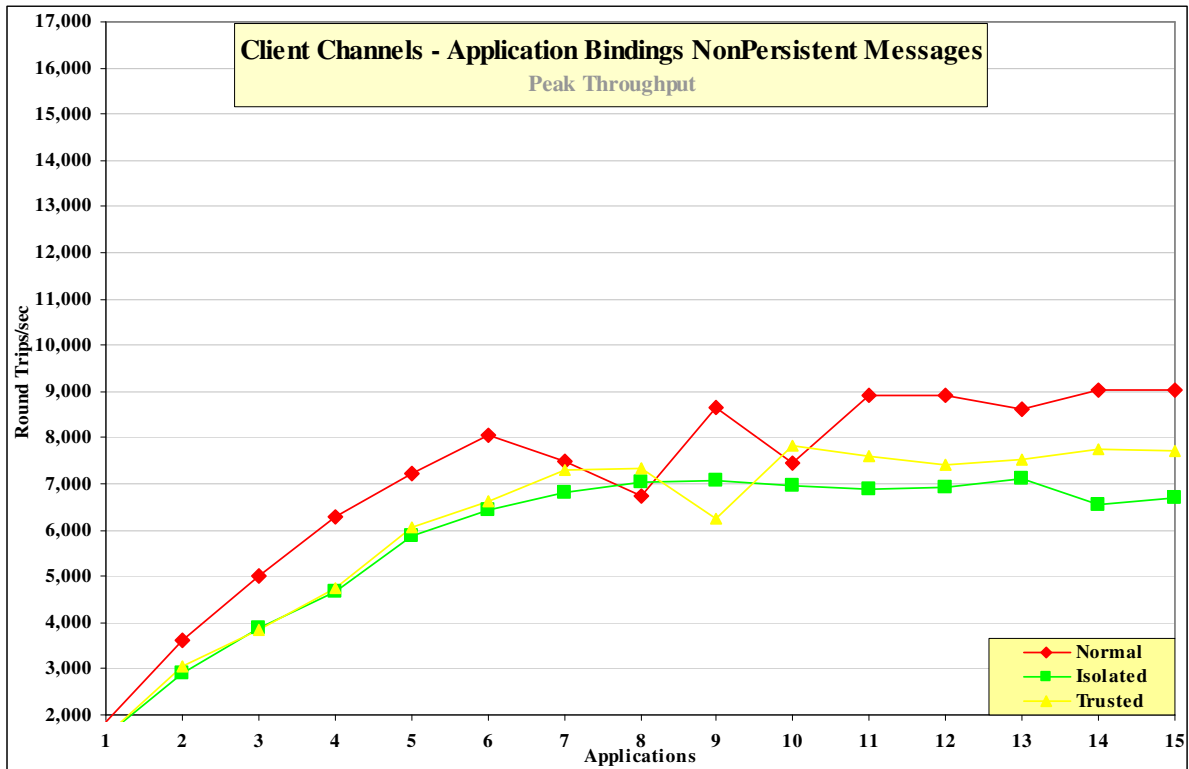


Figure 40 – Application binding, nonpersistent messages, client channels

Figure 40 and Table 29 show that the peak throughput of nonpersistent messages when comparing Normal, Isolated and Trusted bindings.

Test	Apps	Round Trips/sec	Response time (s)	CPU
Normal	14	9018	0.003	94%
Isolated	13	7094	0.003	94%
Trusted	10	7826	0.002	86%

Table 29 – Application binding, nonpersistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The numbers in brackets are included in the table to provide meaningful comparison between WebSphere MQ V6.0 and Version 7.

4.2.2 Persistent Messages

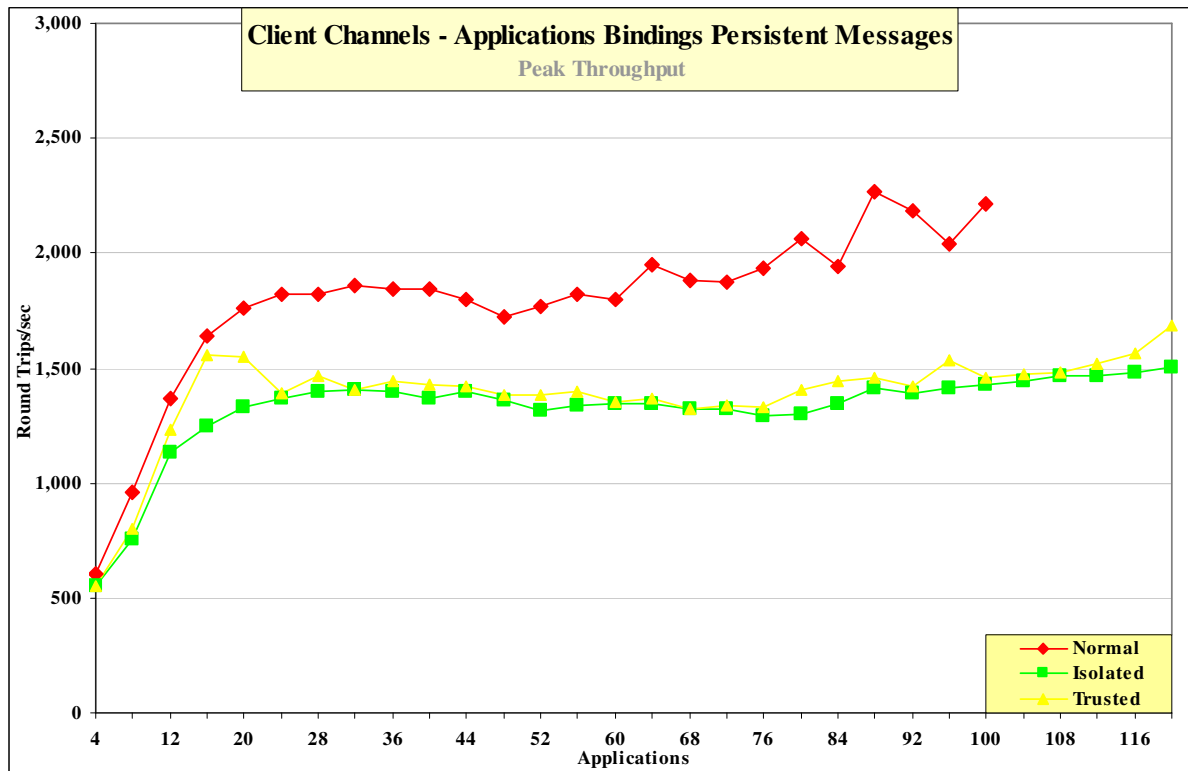


Figure 41 – Application binding, persistent messages, client channels

Figure 41 and Table 30 show that the peak throughput of nonpersistent messages when comparing Normal, Isolated and Trusted bindings.

Test	Apps	Round Trips/sec	Response time (s)	CPU
Normal	100	2215	0.062	88%
Isolated	120	1500	0.094	36%
Trusted	120	1683	0.084	37%

Table 30 – Application binding, persistent messages, client channels

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput.

4.3 Distributed Queuing

Figure 41 and Figure 42 show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario.

4.3.1 Nonpersistent Messages

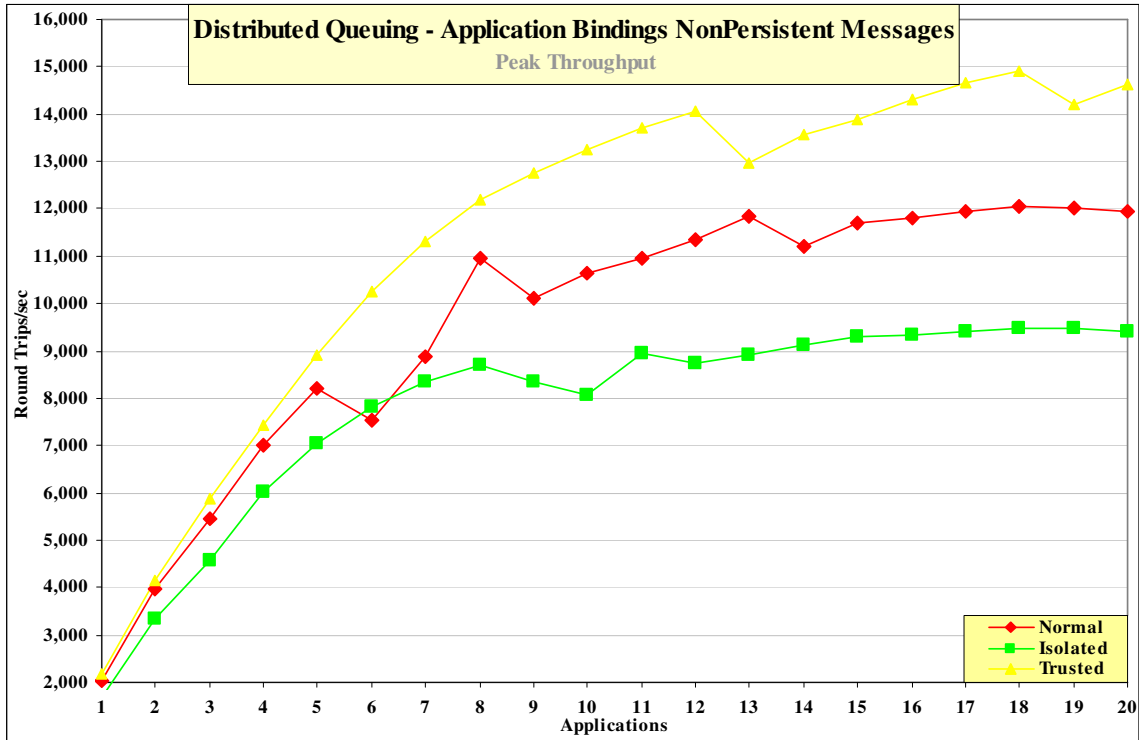


Figure 42 – Application binding, nonpersistent messages, distributed queuing

Figure 42 and Table 31 show that the peak throughput of nonpersistent messages when comparing Normal, Isolated and Trusted bindings.

Test	Apps	Round Trips/sec	Response time (s)	CPU
Normal	18	12042	0.002	92%
Isolated	(18)	(9462)	(0.003)	(75%)
Trusted	18	14904	0.002	80%

Table 31 – Application binding, nonpersistent messages, distributed queuing

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput.

4.3.2 Persistent Messages

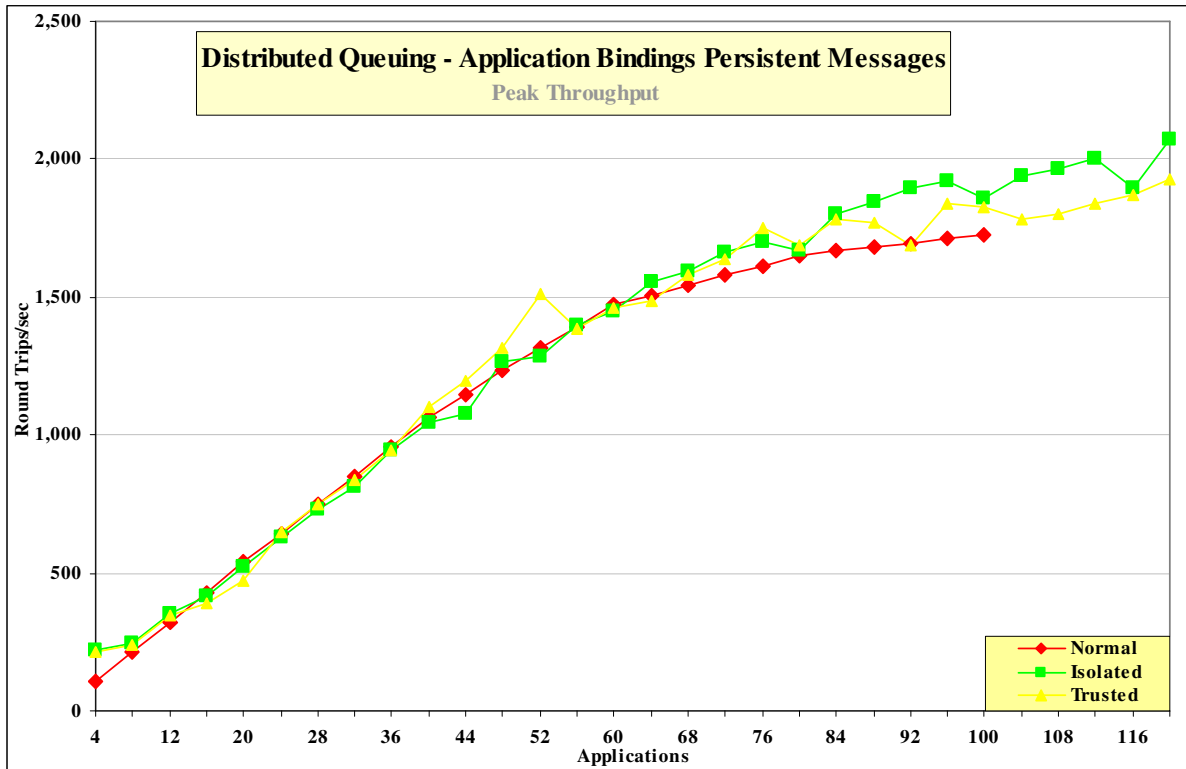


Figure 43 – Application binding, persistent messages, distributed queuing

Figure 43 and Table 32 show that the peak throughput of nonpersistent messages when comparing Normal, Isolated and Trusted bindings. The ‘Normal’ measurement used 300 Requesters to achieve 2747 Round Trips.

Test	Apps	Round Trips/sec	Response time (s)	CPU
Normal	120	1793	0.070	28%
	300	2747	0.128	31%
Isolated	120	2069	0.077	24%
Trusted	120	1928	0.063	18%

Table 32 – Application binding, persistent messages, distributed queuing

Note: The large bold numbers in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput.

5 Short & Long Sessions

The previous chapters in this report only reported on steady state messaging that does not include any session setup and termination function. This chapter specifically bracket groups of five MQPut/MQGet pairs with MQConn/MQDisc and MQOpen/MQClose calls so a comparison of this overhead can be seen.

A short session is a term used to describe the behaviour of an MQI application as it processes a small number of messages using one or more queues and a queue manager. The measurements in this document use an MQI-client application and the following sequence:

- connects to the queue manager
- opens the common input queue, and common reply queue
- puts a request message to the common input queue
- gets the reply message from the common reply queue
- wait one second
- closes both queues
- disconnects from the queue manager



“Why measure short sessions?”

For each new connecting application or disconnecting application, the queue manager and Operating System must start a new process or thread and set up the new connection. As the number of connecting and disconnecting applications increases, the Operating System and queue manager are subjected to a higher load. While these requests are being serviced, the queue manager has less time available to process messages, so fewer driving applications can be reconnected to the queue manager per second before the response time exceeds one second.

This effect is greater than that of reducing the total messaging throughput of the queue manager by connecting thousands of MQI applications to the queue manager

Test name	Apps	Round Trips/sec	Short Sessions per second	Response Time (s)	CPU%
clnp_r3600	3950	3945		0.06	67%
clnp_ss (SC1)	790	768	153	0.17	54%
clpm_r3600	1440	1440		0.15	47%
clpm_ss (SC1)	600	588	117	0.36	58%

Table 33 – Short sessions, client channels

Note: Messaging in these tests is 1 round trip per driving application per second, i.e. 1 short session per driving application every 5 seconds

Note: The figures for non-persistent short sessions were generated with all message processing within sync-point control. All other non-persistent messages within this report were generated outside sync-point control.

The ‘runmqsr’ has a much smaller overhead of connecting to and disconnecting from the queue manager because it only uses a single thread per connection rather than an entire process. INETD listener has a significantly smaller capacity because of the need to create a new process for every client.

6 Performance and Capacity Limits

6.1 Client channels – capacity measurements

The measurements in this section are intended to test the maximum number of client channels into a server queue managers with a messaging rate of 1 round trip per client channel per *minute*. The maximum number of connected applications is likely to be determined by other criteria such as recovery time or manageability. Measurements are also made with smaller number of Client channels where the message insertion rate is increased until the system gets congested. This information is intended to be useful to the reader sizing a system with similar scenarios.

Queue manager configuration for client channels capacity tests:

MaxChannels=50000 (100,000 for clnp_cmax)

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
clnp	13	n/a	8636	0.002	72%
clnp_r3600	3950	3600	3945	0.06	67%
clnp_c6000 (SC1)	6000	1830	3073	0.131	70%
clnp_cmax (SC1)	18200	60	300	0.012	(61%)
cl_persist_c6000(SC1)	6000	450	741	0.24	56%

Table 34 Client Capacity, throughput

* *There was no delay between the response to the previous message and the insertion of the next message with 13 clients.*

The maximum message throughput is achieved when there are a small number of requester applications. The clnp_3600 measurement peaks when the queue of input messages waiting to be processed by the Server application builds up because the server application threads can no longer keep up with the demand. Although this ensures the server threads are always busy, the messages are being spilt from the Queue buffer to the file system and possibly to the disk. Each client uses a thread in the AMQRRMPA processes and the management of lots of threads and lots memory objects results in a larger CPU cost to handle each message. The clnp_cmax_ results show sharecnv=1 because this reflects all of the clients using a separate IP socket. The following table shows how the amount of Available memory and Committed memory is affected by an increased number of clients.

Test name:	Apps	Availbytes	CommBytes
clnp_cmax (SC1)	1000	3227M	1186M
	12700	854M	5497M
	18200	-	-
clnp_c6000(SC1)	1000	3132M	54M
	6000	2393M	1072M
		148K per client	204K per client
cl_persist_c6000(SC1)	1000	3292M	11M
	6000	2515M	1115M
		155K per client	221K per client

Table 35 – Client capacity, memory utilisation

Note: The table above show the AvailBytes and CommBytes measured at the given number of driving applications.

The difference between this pair of measurements is the clnp_cmax uses a Get by Correlation_Id from a common reply queue for all the clients whereas the other case has a separate reply queue per client. Each additional Client needs a thread in the AMQRMPPA process and this accounts for most of the 256Kbytes. Using a separate queue per client needs additional shared memory per client as well as some more pages from

the Free pool. The FREEMEM information is collected at low message rates. An additional 20K per client is needed at the higher rates.

6.2 Distributed queuing – capacity measurements

The measurements in this section are intended to test the maximum number of server channel pairs between two queue managers with a messaging rate of 1 round trip per server channel per *minute*. For the same number of server channel pairs, a faster message rate gives a higher total message throughput over each channel pair. This information is intended to be useful to the reader sizing a system with similar scenarios.

Queue manager and log configuration for distributed queuing capacity tests:

MaxChannels=20000, LogPrimaryFiles=12, LogFilePages=16384, LogBufferPages=512

Note: The large log capacity for this test is for writing the object definitions to the log disk (the transmission queue definitions for both sides of the server channel pair, and reply queue per receiver channel on the driving machine).

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
dqnp	20	n/a*	10987	0.003	93%
dqnp_r3600	5000	3600	4998	0.009	42%
dq_persist_r3600	2900	3600	2699	0.908	37%
dqnp_q1000	1000	18520	4150	0.05	60%
dqnp_qmax	3350	60	56	0.02	

Table 36 – Capacity measurements, server channels

* *There was no delay between the response to the previous message and the insertion of the next message with 20 driving applications..*

The dqnp and dqnp_r3600 both used a total of 4 pairs of Sender/Receiver pairs of channels between queue managers while the dqnp_q1000 and dqnp_qmax used a pair of channels per application. The dqnp_q1000 shows the reduced throughput experienced when 1000 queue managers are connected into a central hub and the following table shows the storage on the central hub.

Test name:	Apps	Availbytes	CommBytes	AvailBytes per Application	CommBytes per Application
dqnp_q1000	100	3098M	80M	417K bytes	431k
	1000	2722M	468M		

Table 37 – DQ capacity, memory utilisation

Note: The table above show the AvailBytes and the CommBytes measured at the given number of driving applications.

7 Tuning Recommendations

7.1 Tuning the Queue Manager

This section highlights the tuning activities that are known to give performance benefits for WebSphere MQ V7.0; some of these can be applied to Version 6. The reader should note that the following tuning recommendations **may not necessarily need** to be applied, especially if the message throughput and/or response time of the queue manager system already meets the required level. Some tuning recommendations that follow may degrade the performance of a previously balanced system if applied inappropriately. The reader should carefully monitor the results of tuning the queue manager to be satisfied that there have been no adverse effects.

Customers should test that any changes have not used excessive real resources in their environment and make only essential changes. For example, allocating several megabytes for multiple queues reduces the amount of shared and virtual memory available for other subsystems, as well as over committing real storage.

Note: The 'TuningParameters' stanza is not documented external interface and may change or be removed in future releases.

7.1.1 Queue Disk, Log Disk, and Message Persistence

Nonpersistent messages are held in main memory, spilt to the file system as the queues become deep, and lazily written to the Queue file. Persistent messages are synchronously written to the log by an MQCmit and also periodically flushed to the Queue file.

To avoid potential queue and log I/O contention due to the queue manager simultaneously updating a queue file and log extent on the same disk, it is important that queues and logs are located on *separate* and *dedicated* physical devices. Multiple disks can be redirected to a Storage Area Network (SAN) but multiple high volume Queue managers can require different Logical Volumes to avoid congestion.

With the queue and log disks configured in this manner, careful consideration must still be given to message persistence: persistent messages should only be used if the message needs to survive a queue manager restart (forced by the administrator or as the result of a power failure, communications failure, or hardware failure). In guaranteeing the recoverability of persistent messages, the pathlength through the queue manager is three times longer than for a nonpersistent message. This overhead does not include the additional time for the message to be written to the log, although this can be minimised by using cached disks or SAN.

7.1.1.1 Nonpersistent and Persistent Queue Buffer

The default nonpersistent queue buffer size is 64K per queue and the default persistent is 128K per queue for 32 bit Queue Managers and 128K /256K for 64 bit Queue Managers (AIX, Solaris, HPUX, Linux_64, z_Linux, and Windows64). They can all be increased to 1MB using the TuningParameters stanza and the *DefaultQBufferSize* and *DefaultPQBufferSize* parameters. (For more details see SupportPac MP01: MQSeries – Tuning Queue Limits). Increasing the queue buffer provides the capability to absorb peaks in message throughput at the expense of real storage. Once these queue buffers are full, the additional message data is given to the file system that will eventually find its way to the disk. Defining queues using large nonpersistent or persistent queue buffers can degrade performance if the system is short of real memory either because a large number of queues have already been defined with large buffers, or for other reasons -- e.g. large number of channels defined.

Note: The queue buffers are allocated in shared storage so consideration must be given to whether the agent process or application process has the memory addressability for all the required shared memory segments.

Queues can be defined with different values of *DefaultQBufferSize* and *DefaultPQBufferSize*. The value is taken from the TuningParameters stanza in use by the queue manager when the queue was defined. When the queue manager is restarted existing queues will keep their earlier definitions and new queues will be created with the current setting. When a queue is opened, resources are allocated according to the definition held on disk from when the queue was created.

7.1.2 Log Buffer Size, Log File Size, and Number of Log Extents

The Log component is often the bottleneck when processing persistent messages. Sufficient information is stored on the log to restart the queue manager after failure. Circular logging is sufficient to recover from application, software, or power failure while linear logging will also recover from media (or disk) failure. Log

records are written at each MQPut, MQGet, and MQCmit into the log buffer. This information is moved onto the log disk. Periodically the Checkpoint process will decide how many of these logfile extents are in the Active log and need to be kept online for recovery purposes. Those extents no longer in the active log are available for archiving when using Linear logging or available for reuse when using circular. There should be sufficient Primary logs to hold the Active log plus the new log extents used until the next checkpoint otherwise some Secondary logs are temporarily included in the log set and they have to be instantly formatted which is an unnecessary delay when using circular logging.

The log buffer is a circular piece of main memory where the log records are concatenated so that multiple log records can be written to the log file in a single I/O operation. The default values used for *LogBufferPages* and *LogFilePages* have been increased in V7 and are probably suitable for most installations. The default size of the log buffer is 512 pages (V6 was 128 pages) with a maximum size of 4096 pages. To improve persistent message throughput of large messages (messages size > 1M bytes) the *LogBufferPages* could be increased to improve likelihood of messages only needing one I/O to get to the disk. Environments that process under 100 small (< 10K byte messages) Persistent messages per second can reduce the memory footprint by using smaller values like 32 pages without impacting throughput. *LogFilePages* (i.e. `crtmqm -lf <LogFilePages>`) defines the size of one physical disk extent (default 4096 pages whereas the V6 default was 128 on Windows and 1024 on Unix). The larger the disk extent, the longer the elapsed times between changing disk extents. It is better to have a smaller number of large extents but long running UOW can prevent Checkpointing efficiently freeing the disk extent for reuse. The largest size (maximum 65536 pages) will reduce the frequency of switching extents. The number of *LogPrimaryFiles* (i.e. `crtmqm -lp <LogPrimaryFiles>`) can be configured to a large number and the maximum number of Primary plus Secondary extents is 255(Windows) and 511(UNIX) but it is for functional reasons rather than performance that need more than 20 primary extents for Circular logging. Circular logging should be satisfied by Primary logs because Secondary logs are formatted each time they are reused. The Active log set is the number of extents that are identified by the Checkpoint process as being necessary to be kept online. As additional messages are processed, more space is taken by the active log. As UOWs complete, they enable the next Checkpoint process to free up extents that now become available for archiving with Linear logging. Some installation will use Linear logging and not archive the redundant logs because archiving impacts the run time performance of logging. They will periodically (daily or twice daily) use 'rcdmqimg' on the main queues thus moving the 'point of recovery' forward, compacting the queues, and freeing up log disk extents. The cumulative effect of this tuning will:

- Improve the throughput of persistent messages (enabling by default a possible 2MB of log records to be written from the log buffer to the log disk in a single write). Initial target - half to one second of log datastreaming into the Logbuffer.
- Reduce the frequency of log switching (permitting a greater amount of log data to be written into one extent). Initial target - LogFile extent hold at least 10 seconds of log datastreaming.
- Allow more time to prepare new linear logs or recycle old circular logs (especially important for long-running units of work).

Changes to the queue manager *LogBufferPages* stanza take effect at the next queue manager restart. The number of pages can be changed for all subsequent queue managers by changing the *LogBufferPages* parameter in the product default Log stanza.

It is unlikely that poor persistent message throughput will be attributed to a 2MB queue manager log but processing of large messages will be helped by these enhanced limits. It is possible to fill and empty the log buffer several times each second and reach a CPU limit writing data into the log buffer, before a log disk bandwidth limit is reached.

7.1.2.1 LogWriteIntegrity: SingleWrite or TripleWrite

The default value is TripleWrite. MQ writes log records using the TripleWrite method because it provides full write integrity where hardware that assures write integrity is not available. Some hardware guarantees that, if a write operation writes a page and fails for any reason, a subsequent read of the same page into a buffer results in each byte in the buffer being either:

- The same as before the write, or
- The byte that should have been written in the write operation

On this type of hardware (for example, SSA write cache enabled), it is safe for the logger to write log records in a single write as the hardware assures full write integrity. This method provides the highest level of performance.

Queue manager workloads that have multiple streams asynchronously creating high volume log records will not benefit from 'SingleWrite' because the logger will not need to rewrite partial pages

of the log file. Workloads that serialize on a small number of threads where the response time from an MQGet, MQPut, or MQCmit inhibits the system throughput are likely to benefit from Singlewrite and could enhance throughput by 25%.

7.1.3 Channels: Process or Thread, Standard or Fastpath?

Threaded channels are used for all the measurements in this report ('runmqslr', and for server channels an MCATYPE of 'THREAD') the threaded listener 'runmqslr' can now be used in all scenarios with client and server channels. Additional resource savings are available using the 'runmqslr' listener rather than 'inetd', including a reduced requirement on: virtual memory, number of processes, file handles, and System V IPC.

Fastpath channels, and/or fastpath applications—see later paragraph for further discussion, can increase throughput for both nonpersistent and persistent messaging. For persistent messages, the improvement is only for the path through the queue manager, and does not affect performance writing to the log disk.

Note: The reader should note that since the greater proportion of time for persistent messages is in the queue manager writing to the log disk, the performance improvement for fastpath channels is less apparent with persistent messages than with nonpersistent messages.

7.1.4 Multiplexed clients

Version 6 and previous levels used a separate TCP socket for each client. Version 7 will multiplex clients from the same process over one TCP socket. Chapter 2 show the difference in performance of these variants. Version 6 behaviour can be obtained by using the 'sharecnv' keyword with a setting of zero. For example

```
define channel( csim_channel_TCP ) +
    chltype( svrconn ) +
    trptype( tcp ) +
    sharecnv( 0 )
```

Version 6 behaviour will also inhibit new performance features of V7 like 'ASYNc Put and 'READ_AHEAD'. This is referred to as Compatibility mode.

7.2 Applications: Design and Configuration

7.2.1 Standard (Shared or Isolated) or Fastpath?

The reader should be aware of the issues associated with writing and using fastpath applications—described in the 'MQSeries Application Programming Guide'. Although it is recommended that customers use fastpath channels, it is not recommended to use fastpath applications. If the performance gain offered by running fastpath is not achievable by other means, it is essential that applications are rigorously tested running fastpath, and never forcibly terminated (i.e. the application should always disconnect from the queue manager). Fastpath channels are documented in the 'MQSeries Intercommunication Guide'.

7.2.2 Parallelism, Batching, and Triggering

An application should be designed wherever possible to have the capability to run *multiple instances* or *multiple threads* of execution. Although the capacity of a multi-processor (SMP) system can be fully utilised with a small number of applications using nonpersistent messages, more applications are typically required if the workload is mainly using persistent messages. Processing messages inside syncpoint can help reduce the amount of time the queue managers takes to write a group of persistent messages to the log disk. The performance profile of a workload will also be subject to variability through cycles of low and heavy message volumes, therefore a degree of experimentation will be required to determine an optimum configuration.

Queue avoidance is a feature of the queue manager that allows messages to be passed directly from an 'MQPUTer' to an 'MQGETer' without the message being placed on a queue. This feature only applies for processing messages outside of syncpoint. In addition to improving the performance of a workload with multiple parallel applications, the design should attempt to ensure that an application or application thread is always available to process messages on a queue (i.e. an 'MQGETer'), then messages outside of syncpoint do not need to ever be physically placed on a queue.

The reader should note that as more applications are processing messages on a single queue there is an increasing likelihood that queue avoidance will not be maintainable. The reasons for this have a cumulative and exponential effect, for example, when messages are being placed on a queue quicker than they can be removed. The first effect is that messages begin to fill the queue buffer—and MQGETers need to retrieve messages from the buffer rather than being received directly from an MQPUTer. A secondary effect is that as messages are spilled from the buffer to the queue disk, the MQGETers must wait for the queue manager to retrieve the message from the queue disk rather than being retrieved from the queue buffer. While these problems can be addressed by configuring for more MQGETers (i.e. processing threads in the server application), or using a larger queue buffer, it may not be possible to avoid a performance degradation.

Processing persistent messages inside syncpoint (i.e. in batches) can be more efficient than outside of syncpoint. As the number of messages in the batch increases, the average processing cost of each message decreases. For persistent messages the queue manager can write the entire batch of messages to the log disk in one go while outside of syncpoint control, the queue manager must wait for each message to be written to the log before returning control to the application.

Only one log record per queue can be written to the disk per log I/O when processing messages outside of syncpoint. This is not a bottleneck when there are a lot of different queues being processed. When there are a small number of queues being processed by a large number of parallel application threads, it is a bottleneck. By changing all the messages to be processed inside syncpoint, the bottleneck is removed because multiple log records per queue can share the same log I/O for messages processed within syncpoint.

A typical triggered application follows the performance profile of a short session. The 'runmqslr' has a much smaller overhead of connecting to and disconnecting from the queue manager because it does not have to create a new process. The programmatical implementation of triggering is still worth consideration with regard to programming a disconnect interval as an input parameter to the application program. This can provide the flexibility to make tuning adjustments in a production environment, if for instance, it is more efficient to remain connected to the queue manager between periods of message processing, or disconnect to free queue manager and Operating System resources.

7.3 Virtual Memory, Real Memory, & Paging

Systems require sufficient real memory to hold the working set otherwise paging will break the response time expectations.

- Virtual memory enables the program to address much larger amount of memory than exists as real memory.
- Real memory is the physical memory (or RAM) currently installed in the machine.
- Paging is the process of managing program access to virtual storage pages not currently resident in main memory. It locates the required page frame from auxiliary storage (disk), selects a page frame in real memory that will hold this page, copies the contents of this outgoing page frame to auxiliary storage, and retrieves the requested incoming page contents from auxiliary storage.

A simple approach is to ensure that the virtual memory of the application system does not exceed the available real memory since all memory requests will be met from the current free memory.

WebSphere MQ uses a significant amount of memory for each Queue Manager and Channel.

7.3.1 Queue Manager

Starting a MQ Queue manager generated using default values increases the AVM and reduces the FRE by 32M bytes.

7.3.2 Channels

Channels can be started by using the INETD or the RUNMQLSR listener. INETD initiated channels use between 5 and 10 times more memory than RUNMQLSR channels so the rest of this section focuses on RUNMQLSR channels.

7.3.3 Client Channels

Each MQ client channel uses between 264K - 410K bytes for processing 2K byte messages depending on traffic rate (Chapter 6 of the MQ V6 Performance reports provides an estimate of the storage needed when clients either share a predefined queue with other clients or have a dynamic queue per client). 100K byte messages will use up to 700K bytes per client.

7.3.4 Server – Server Channels

Each interconnected queue manager has a pair of uni-directional channels for sending and receiving messages. The storage consumed is the same as 2 client channels plus a predefined queue (Transmission queue).

Three other aspects of storage consumption depend on type of 'Reply-Queue', MQIBINDTYPE, and BufferLength.

7.3.5 Reply Queue

The Queue from which the client retrieves the message can be a predefined Queue (350K bytes) probably shared among multiple clients who get messages by Correlation-id or a model (dynamic) queue (60K bytes) that is used only by one client. The model queue memory can grow by 128K bytes when more than 128K bytes of Persistent messages are held in the queue and by 192K bytes when more than 192K bytes of non persistent messages are held in the queue. This memory is not shrunk back to the underlying 60K bytes for model queues.

7.3.6 BufferLength

The AMQRMPPA process contains a thread per connected client. The BufferLength parameter of the MQGet is also used to allocate a long term piece of storage of this size in which the message is held before being retrieved by the client. If the size of the arriving messages cannot be predicted then the application should provide a buffer that can deal with 90% of the messages and redrive the MQGET after return code **2080 (X'0820') MQRC_TRUNCATED_MSG_FAILED** by providing a larger BUFFER for retrieving this particular message. There is a mechanism to gradually reduce the size of the storage in AMQRMPPA if the recent BufferLength size is significantly smaller than previous BufferLength.

7.3.7 MQIBINDTYPE

MQIBINDTYPE=FASTPATH will cause the channel to run 'Trusted' mode. Trusted applications do not use a thread in the Agent (AMQZLLA) process. This means there is no IPC between the Channel and Agent because the Agent does not exist in this connection. If the channel is run in STANDARD mode then any messages passed between the channel and agent will use IPCC memory (size = BufferSize with a maximum size of 1MB) that is dynamically obtained and only held for the lifetime of the MQGet. Standard channels each require an additional 80K bytes of memory. As the message rate increases, there will be more IPCC memory used in parallel.

The power of the machine used to process a workload needs to handle the peaks of troughs. Customers may specify a daily workload but this number cannot be divided by the number of seconds in a day to find the necessary system configuration. The peak hourly rate cannot be divided by 3600 because the peak rate per second will probably be 2-3 times higher. The system must process these peak loads without building up a backlog of queued work. It is important to prevent the queue depths increasing because they will occupy memory from the 'fre' pool or be spilled out to disk. Over commitment of real memory is handled by the page manager but sudden large jumps (storms) possibly due to queues becoming deep can cause the throughput to break down completely if the page manager chooses too much working set memory to be paged. Gradual over commitment enables the page manager to shuffle out those pages that are not part of the working set.

7.4 TCP/IP tuning for capacity number

When a client makes a connect() call to make a connection to a server, the default range for dynamic tcp/ip ports in Windows is 1024 to 5000, thus giving 3977 outbound concurrent connections for each IP Address. It is possible to change the upper limit with this DWORD registry key:

```
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters]
MaxUserPort = 5000 (Default = 5000, Max = 65534)
```

This is required when running our capacity numbers where we attempt to connect 6000-max applications

8 Measurement Environment

8.1 Workload description

8.1.1 MQI response time tool

The MQI tool exercises the local queue manager by measuring elapsed times of the 8 main MQSeries verbs: MQCONN(X), MQDISC, MQOPEN, MQCLOSE, MQPUT, MQGET, MQCMIT, and MQBACK. The following MQI calls are paired together inside a test application:

- MQCONN(X) with MQDISC
- MQOPEN with MQCLOSE
- MQPUT with MQGET
- MQCMIT and MQBACK with MQPUT and MQGET

Note: MQCLOSE elapsed time is only measured for an empty queue.

Note: Performance of MQCMIT and MQBACK is measured in conjunction with MQPUT and MQGET, putting and getting messages inside a unit of work (i.e. inside syncpoint control). The unit of work is committed at the end of each batch. The number of messages per batch is a parameter of the test.

Note: This tool is not used to measure the performance of verbs: MQSET, MQINQ, or MQBEGIN.

8.1.2 Test scenario workload

The MQI applications use 64 bit libraries for MQ V6 & V7

8.1.2.1 The driving application programs

The test scenario workload simulates many driving applications running on a single driving machine. This is not typical of a customer environment and is only used to facilitate test coordination. Driving applications were multi-threaded with each thread performing a sequence of MQI calls. The driving applications (Requesters) for Local and DQ tests used Trusted bindings. The number of threads in each application was adjusted according to whether the test was measuring a local queue manager, a client channel, or distributed queuing scenario. This was done to reduce storage overheads on the driving system. Each driving application thread performed the sequence of actions as outlined in the test scenario illustrations in the ‘**Performance Headlines**’ starting on **page 2**.

Message rate: in all but the *rated* and *capacity limit* tests, message processing was performed in a *tight-loop*. In the *rated* tests a message rate of 1 round trip per driving application per *second* was used, and in the *capacity limit* tests a message rate of 1 round trip per channel per *minute* was used.

Nonpersistent and persistent messages were used in all but the *capacity limit* tests.

Note: The driving applications gathered timing information for all MQI calls using a high-resolution timer.

8.1.2.2 The server application program

The server application is written as a multi-threaded program configured to use 20, 6, 6 threads for processing nonpersistent messages with Local, Client, and DQ applications, and 30, 60, 10 threads to process persistent messages with Local, Client, and DQ applications. The capacity tests in chapter 5 and 6 use 10 server threads for processing non persistent messages. Each server thread performed the sequence of actions as outlined in the test scenario illustrations in the “**Performance Headlines**” starting on **page 2**.

Nonpersistent messaging is done outside of syncpoint control. Persistent messaging is done inside of syncpoint control. The average message throughput expressed as a number of round trips per second was calculated and reported by the server program.

8.2 Hardware

IBM x3850: Server system (Device under test)
Model: x3850 M2
Processor: 3.3GHz Intel xeon
Architecture: 2 Dual core cpus (4 way SMP)
Memory (RAM): 4GB
Disk: 2 Internal 16bit SCSI (9GB each, 1 O/S, swap)
2 SAN disks on DS6000 (5GB each, 1 queue, 1 log)
Network: 1GBit Ethernet Adapter

IBM x3850: Driver system
Model: x3850 M2
Processor: 3.3GHz Intel xeon
Architecture: 2 Dual core) cpus (4 way SMP)
Memory (RAM): 4GB
Disk: 2 Internal 16bit SCSI (9GB each, 1 O/S, swap)
2 SAN disks on DS6000 (5GB each, 1 queue, 1 log)
Network: 1GBit Ethernet Adapter

8.3 Software

Windows : MicroSoft Windows Server 2003 Enterprise x64 Edition
MQSeries: Version 7.0, Version 6.0
Compiler: C, Version 6

9 Glossary

Test name	<p>The name of the test.</p> <p><i>Note: The test names in some cases are rather long. This is done to provide a descriptive qualification of the test measurement to relate to the performance discussion in the sections throughout the document:</i></p> <p>local => local queue manager test scenario</p> <p>cl => client channel test scenario</p> <p>dq => distributed queuing test scenario</p> <p>np => nonpersistent messages</p> <p>pm => persistent messages</p> <p>r3600 => 1 round trip per driving application per second</p> <p>runmqtsr => channels using the 'runmqtsr' listener (client channel test scenario, in addition to 'runmqchi' for distributed queuing test scenarios)</p> <p>c6000 => 6,000 client driving applications (i.e. 6,000 MQI-client connections)</p> <p>q1000 => 1,000 server channel pairs</p> <p>max => maximum number of channels (or channel pairs)</p> <p>no_correl_id => correlation identifier not used in the response messages (as each response is placed on a unique reply-to queue per driving application)</p>
Apps	The number of driving applications connected to the queue manager at the point where the performance measurement is given.
Rate/App/hr	The target message throughput rate of each driving application.
Round T/s	The average achieved message throughput rate of all the driving applications together, measured by the server application.
% (Round T/s)	<p>The percentage increase in the total message throughput rate.</p> <p><i>Note: The nature of the comparison is noted under each table where percentage improvements have been given.</i></p>
CPY	As reported by Windows system monitor
Resp time (s)	The average response time each round trip, as measured and averaged by all the driving applications.
CURDEPTH	<p>The number of messages on the input queue as a snapshot.</p> <p><i>Note: runmqsc <qmname>, DISPLAY QLOCAL(<qname>) CURDEPTH</i></p>
queue disk (kbps)	The queue disk kilobytes transferred per second.
Swap	The total amount of swap area reservation for all processes in MB, unless otherwise specified as swap/app (i.e. swap area reservation per driving application).
shm	The amount of allocated shared memory in MB.
FREE	Available memory as reported by Windows system monitor