WebSphere MQ Linux V7.0 - Performance Evaluations on xSeries 64 bit

Version 1.3

June 2010

Peter Toghill , Craig Stirling .

WebSphere MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
SO21 2JN

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.

Forth Edition, June 2010.

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

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 Linux 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 Linux.
- Want to understand the performance limits of WebSphere MQ for Linux V7.0.
- Want to understand what actions may be taken to tune WebSphere MQ for Linux.

The reader should have a general awareness of the Linux operating system and of MQSeries in order to make best use of this SupportPac. Readers should read the section '**How this document is arra**'—**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.

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 2Kb (2,048 bytes).

An xSeries 3850 M2 2 dual core cpu 3.33GHz Intel xeon with 4Gb of RAM was used as the Device under test.

An xSeries 3850 M2 2 dual core cpu 3.33GHz Intel xeon with 4Gb of RAM was used as the Driver.

How this document is arranged

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: 19-40

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 "*WebSphere MQ* V7.0 on Linux has similar performance characteristics to the V6 product. The comparisons in this report show that throughput has dropped by an average of 6% overall (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 13%.

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.

Performance Headlines".

Application Bindings

Page: 49-54

Section three contains performance measurements for *'trusted, normal, and isolated'* server applications, using the same three scenarios as for the "*WebSphere MQ* V7.0 on Linux has similar performance characteristics to the V6 product. The comparisons in this report show that throughput has dropped by an average of 6% overall (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 13%.

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.

Performance Headlines".

Tuning Recommendations

Pages: 59-

Measurement Environment

Pages: 64 66

A summary of the way in which the workload is used in each test scenario is given in the "WebSphere MQ V7.0 on Linux has similar performance characteristics to the V6 product. The comparisons in this report show that throughput has dropped by an average of 6% overall (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 13%.

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.

Performance Headlines" section. This includes a more detailed description of the workload, hardware and software specifications.

Glossary

Page: 66

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
2.3.5 SSL 17
3 Large Messages 19
3.1 MQI Response Times: 50bytes to 100Mb – Local Queue Manager 19
3.1.1 50bytes to 32Kb 19
3.1.2 32Kb to 2Mb 20
3.1.3 2Mb to 100Mb 22
3.2 20K Messages 23
3.2.1 Local Queue Manager 23
3.2.2 Client Channel 25
3.2.3 Distributed Queuing 27
3.3 200K Messages 29
3.3.1 Local Queue Manager 29
3.3.2 Client Channel 31
3.3.3 Distributed Queuing 33
3.4 2Mb Messages 35
3.4.1 Local Queue Manager 35
3.4.2 Client Channel 37
3.4.3 Distributed Queuing 39
3.5 SSL Clients 40
3.5.1 Non Persistent 2K byte message 41
3.5.2 Non Persistent 20K byte message 42
3.5.3 Non Persistent 64K byte message 43
3.5.4 Non Persistent 200K byte message 44
3.5.5 Persistent 2K byte message 45
3.5.6 Persistent 20K byte message 46
3.5.7 Persistent 64K byte message 47
3.5.8 Persistent 200K byte message 48
4 Application Bindings 49
4.1 Local Queue Manager 49
4.1.1 Nonpersistent Messages 49
4.1.2 Persistent Messages 50
4.2 Client Channels 51
4.2.1 Nonpersistent Messages 51
4.2.2 Persistent Messages 52
4.3 Distributed Queuing 53
4.3.1 Nonpersistent Messages 53
4.3.2 Persistent Messages 54
5 Short & Long Sessions 55
6 Performance and Capacity Limits 57
6.1 Client channels – capacity measurements 57
6.2 Distributed queuing – capacity measurements 58
7 Tuning Recommendations 59
7.1 Tuning the Queue Manager 59
7.1.1 Queue Disk, Log Disk, and Message Persistence 59
7.1.2 Log Buffer Size, Log File Size, and Number of Log Extents 59

7.1.3 Channels: Process or Thread, Standard or Fastpath? 61
7.1.4 Multiplexed clients 61
7.2 Applications: Design and Configuration 61
7.2.1 *Standard (Shared or Isolated)* or Fastpath? 61
7.2.2 Parallelism, Batching, and Triggering 61
7.3 Tuning the Operating System (Linux RHES) 62
7.4 Virtual Memory, Real Memory, & Paging 62
7.4.1 Queue Manager 62
7.4.2 Channels 63
7.4.3 Client Channels 63
7.4.4 Server – Server Channels 63
7.4.5 Reply Queue 63
7.4.6 BufferLength 63
7.4.7 MQIBINDTYPE 63
8 Measurement Environment 64
8.1 Workload description 64
8.1.1 MQI response time tool 64
8.1.2 Test scenario workload 64
8.2 Hardware 66
8.3 Software 66
9 Glossary 67

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 23
Table 10 – 20K persistent messages, local queue manager 24
Table 11 – 20K nonpersistent messages, client channels 25
Table 12 – 20K persistent messages, client channels 26
Table 13 – 20K nonpersistent messages, client channels 27
Table 14 – 20K persistent messages, client channels 28
Table 15 – 200K nonpersistent messages, local queue manager 29
Table 16 – 200K persistent messages, local queue manager 30
Table 17 – 200K nonpersistent messages, client channels 31
Table 18 – 200K persistent messages, client channels 32
Table 19 – 200K nonpersistent messages, distributed queuing 33
Table 20 – 200K persistent messages, distributed queuing 34
Table 21 – 2M nonpersistent messages, local queue manager 35
Table 22 – 2M persistent messages, local queue manager 36
Table 23 – 2M nonpersistent messages, client channels 37
Table 24 – 2M persistent messages, client channels 38
Table 25 – 2M nonpersistent messages, distributed queuing 39
Table 26 – 2M persistent messages, distributed queuing 40
Table 27 – Application binding, nonpersistent messages, local queue manager 49
Table 28 – Application binding, persistent messages, local queue manager 50
Table 29 – Application binding, nonpersistent messages, client channels 51
Table 30 – Application binding, persistent messages, client channels 52
Table 31 – Application binding, nonpersistent messages, distributed queuing 53
Table 32 – Application binding, persistent messages, distributed queuing 54
Table 33 – Short sessions, client channels 56
Table 34 – Capacity measurements, client channels 57
Table 35 – Capacity measurements, server channels 58
Table 36 – DQ capacity, memory utilisation 58

FIGURES

Figure 1 – Connections into a local queue manager 2
Figure 2 – Performance headline, nonpersistent messages, local queue manager. 3
Figure 3 NonPersistent nonTrusted 4
Figure 4 – Performance headline, persistent messages, local queue manager 5
Figure 5 – MQI-client channels into a remote queue manager 6
Figure 6 – Performance headline, nonpersistent messages, client channels 7
Figure 7 – Performance headline, persistent messages, client channels 9
Figure 8 – 1 round trip per driving application per second, client channels, nonpersistent messages 10
Figure 9 – 1 round trip per driving application per second, client channels, persistent messages 11
Figure 10 – Server channels between two queue managers 12
Figure 11 – Performance headline, nonpersistent messages, server channels 13
Figure 12 – Performance headline, persistent messages, server channels 15
Figure 13 – 1 round trip per driving application per second, server channel, nonpersistent messages 16
Figure 14 – 1 round trip per driving application per second, server channel, persistent messages 16
Figure 15 –The effect of nonpersistent message size on MQI response time (50byte - 32K) 19
Figure 16 –The effect of persistent message size on MQI response time (50byte - 32K) 19
Figure 17 –The effect of nonpersistent message size on MQI response time (32K – 2Mb) 20
Figure 18 –The effect of persistent message size on MQI response time (32K – 2Mb) 21
Figure 19 –The effect of nonpersistent message size on MQI response time (2Mb – 100Mb) 22
Figure 20 –The effect of persistent message size on MQI response time (2Mb – 100Mb) 22
Figure 21 – 20K nonpersistent messages, local queue manager 23
Figure 22 – 20K persistent messages, local queue manager 24
Figure 23 – 20K nonpersistent messages, client channels 25
Figure 24 – 20K persistent messages, client channels 26
Figure 25 – 20K nonpersistent messages, distributed queuing 27
Figure 26 – 20K persistent messages, distributed queuing 28
Figure 27 – 200K nonpersistent messages, local queue manager 29
Figure 28 – 200K persistent messages, local queue manager 30
Figure 29 – 200K nonpersistent messages, client channels 31
Figure 30 – 200K persistent messages, client channels 32
Figure 31 – 200K nonpersistent messages, distributed queuing 33
Figure 32 – 200K persistent messages, distributed queuing 34
Figure 33 – 2M nonpersistent messages, local queue manager 35
Figure 34 – 2M persistent messages, local queue manager 36
Figure 35 – 2M nonpersistent messages, client channels 37
Figure 36 – 2M persistent messages, client channels 38
Figure 37 – 2M nonpersistent messages, distributed queuing 39
Figure 39 2M persistent messages, distributed queuing 40
Figure 40 2K non-persistent message 41
Figure 41 20K non-persistent message 42
Figure 42 64K Non-persistent message 43
Figure 43 200k Non-persistent message 44
Figure 44 – 2K persistent messages 45
Figure 45 – 20K persistent messages 46
Figure 46 – 64K persistent messages 47
Figure 47 – 200K persistent messages 48
Figure 48 – Application binding, nonpersistent messages, local queue manager 49
Figure 49 – Application binding, persistent messages, local queue manager 50
Figure 50 – Application binding, nonpersistent messages, client channels 51
Figure 51 – Application binding, persistent messages, client channels 52
Figure 52 – Application binding, nonpersistent messages, distributed queuing 53
Figure 53 – Application binding, persistent messages, distributed queuing 54
Figure 54 – Short sessions, client channels 55

1 Overview

WebSphere MQ V7.0 on Linux has similar performance characteristics to the V6 product. The comparisons in this report show that throughput has dropped by an average of 6% overall (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 13%.

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 stopped automatically after the response time exceeds 1 second.

2.1 Local Queue Manager Test Scenario

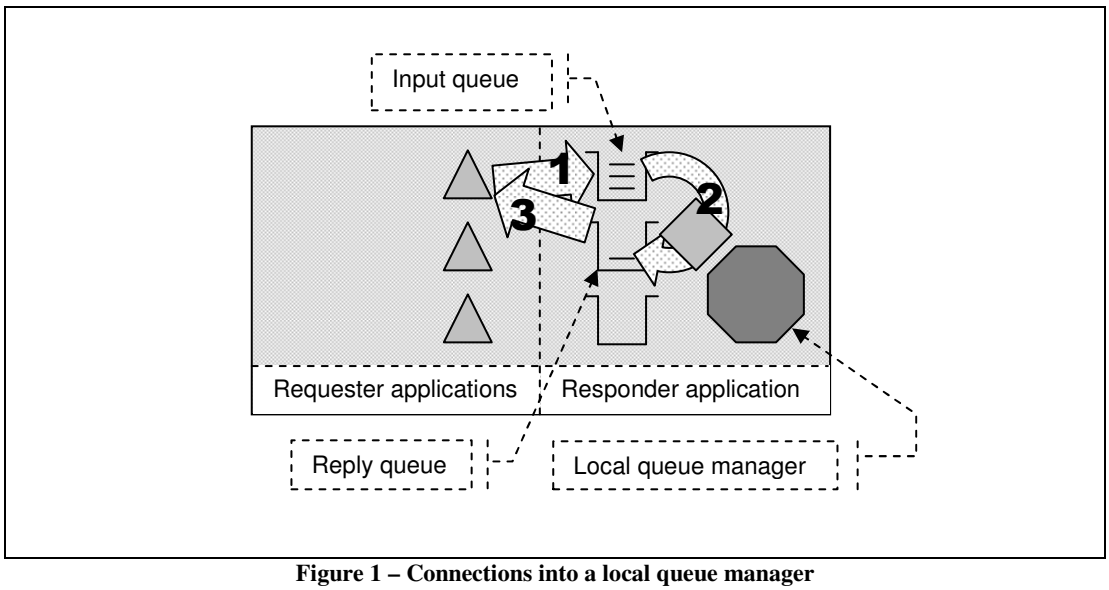


Figure 1 – Connections into a local queue manager

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 3** and **Figure 4** 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.

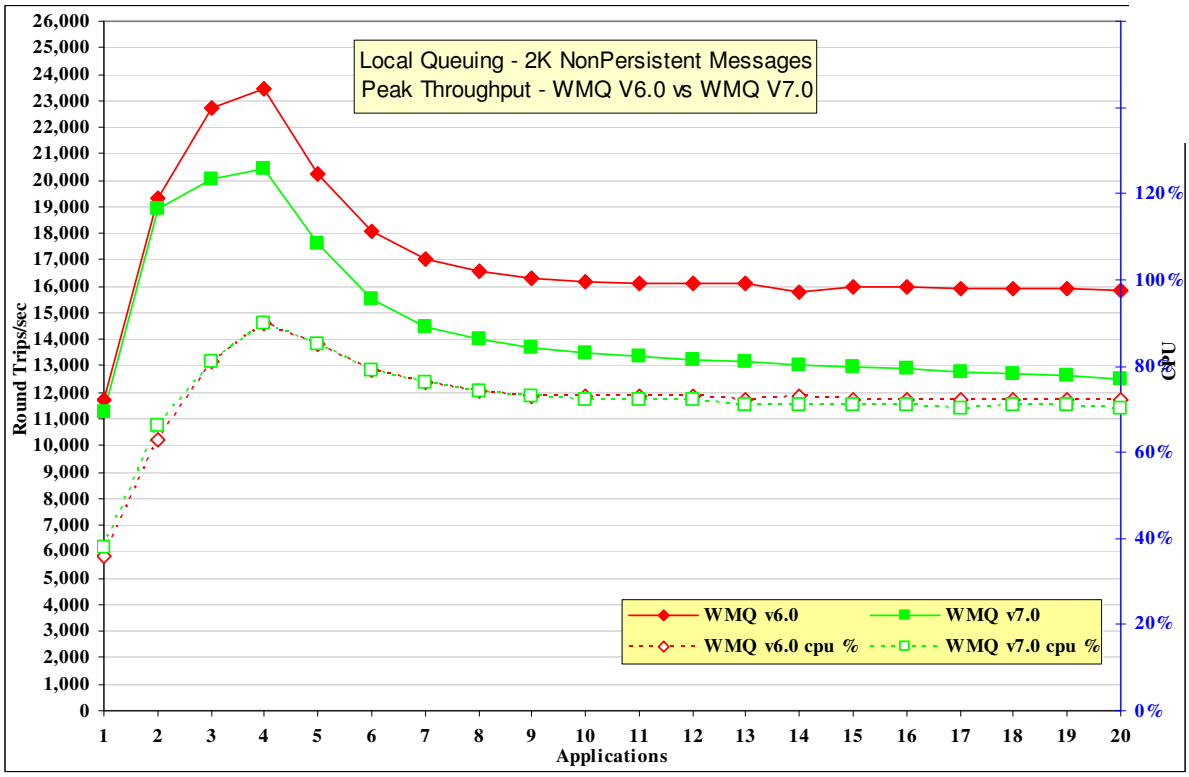


Figure 2 – Performance headline, nonpersistent messages, local queue manager.

Figure 2 and **Table 1** shows that the throughput of nonpersistent messages has reduced by 16% comparing Version 6 to Version 7

Test name: **local_np**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**4**	**23421**	**0.0002**	**90%**
WebSphere MQ V7.0	**4**	**20400**	**0.0003**	**85%**

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

2.1.2 Nonpersistent Messages – Non trusted – Local Queue Manager

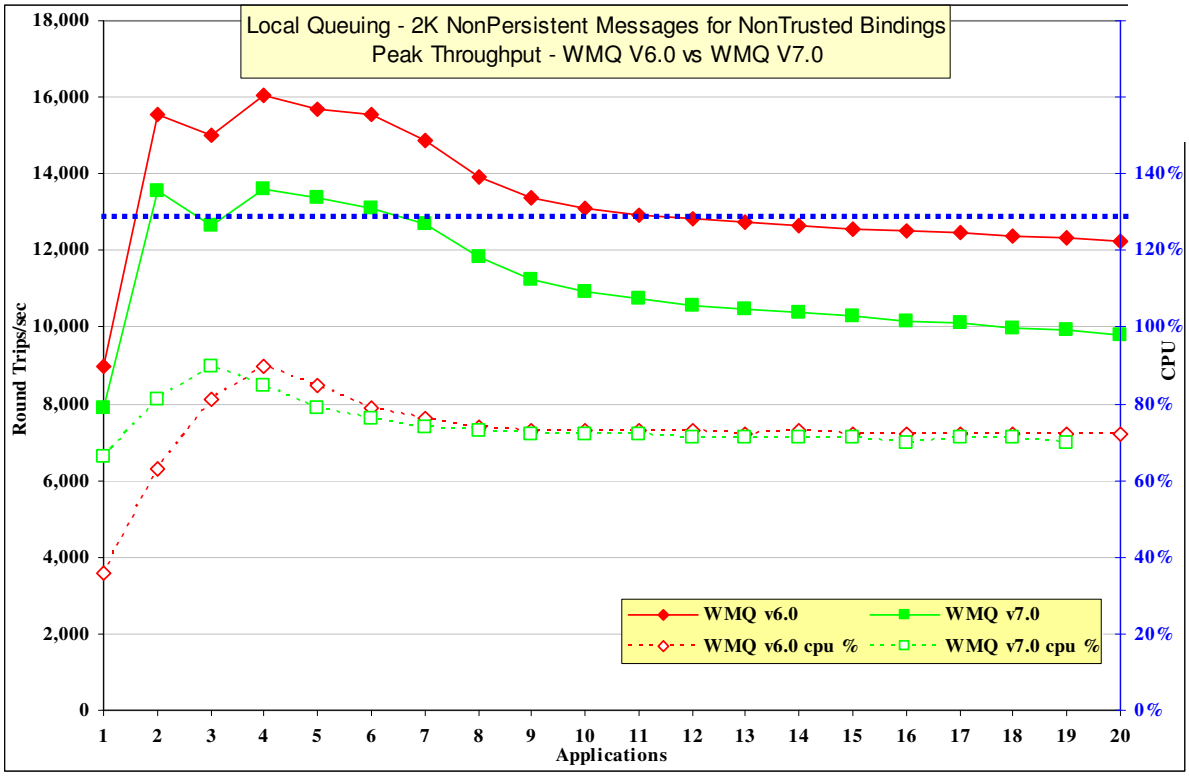


Figure 3 NonPersistent nonTrusted

Test name: **local_np_nt**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**4**	**16060**	**0.0003**	**93%**
WebSphere MQ V7.0	**4**	**13606**	**0.0003**	**94%**

shows that the throughput of nonpersistent messages when the Requester and Responder both use Shared bindings has reduced by 17% comparing Version 6 to Version 7

2.1.3 Persistent Messages – Local Queue Manager

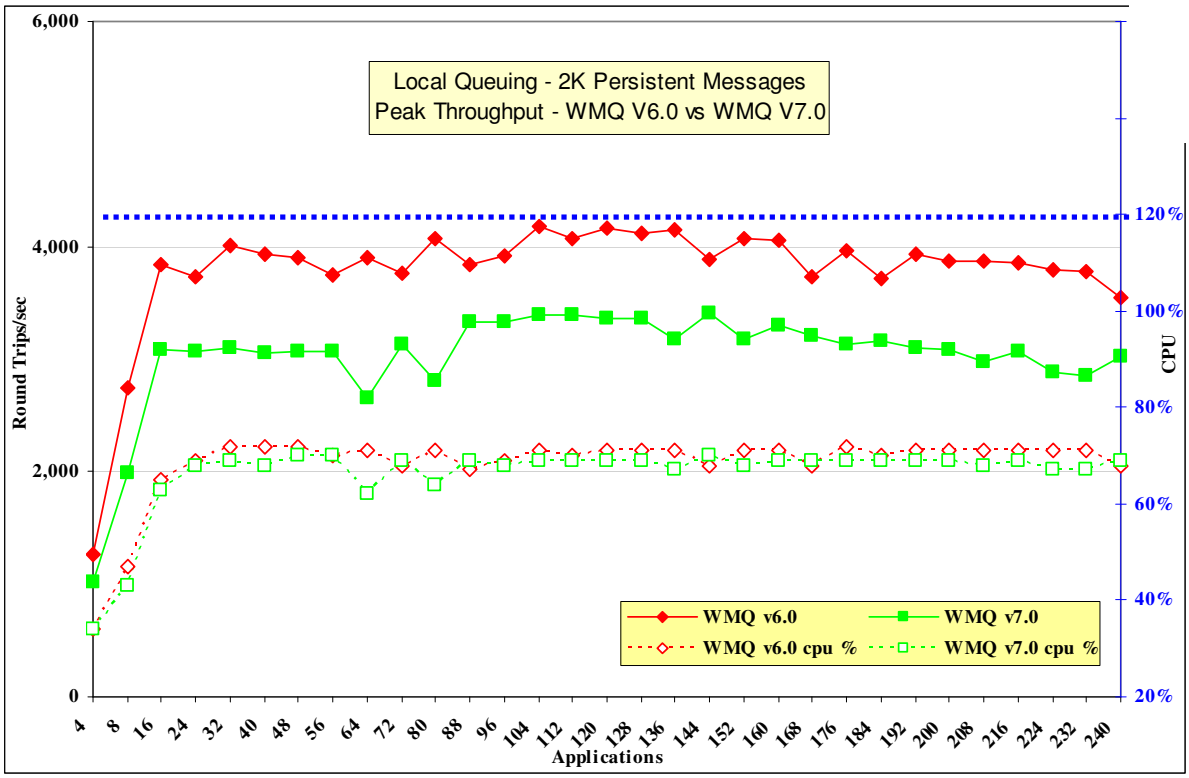


Figure 4 – Performance headline, persistent messages, local queue manager

Figure 4 and **Table 2** show that the throughput of persistent messages has degraded by 20% comparing Version 6 to Version 7.

Test name: **local_pm**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**104**	**4180**	**0.034**	**71%**
WebSphere MQ V7.0	**144**	**3403**	**0.053**	**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

2.2 Client Channels Test Scenario

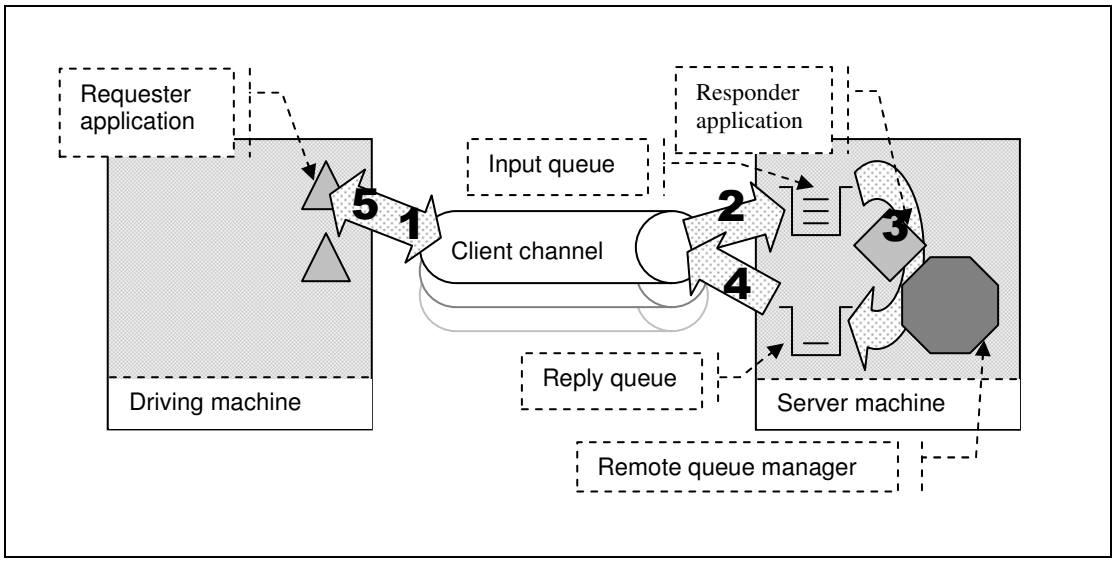


Figure 5 – MQI-client channels into a remote queue manager

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 =STANDARD' is used.

Version 7 will multiplex multiple clients from the same process 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 6 **Figure 6a**, and **Figure 7** shows the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario (see **Figure 5** on the previous page), and WebSphere MQ V7.0 compared to Version 6.

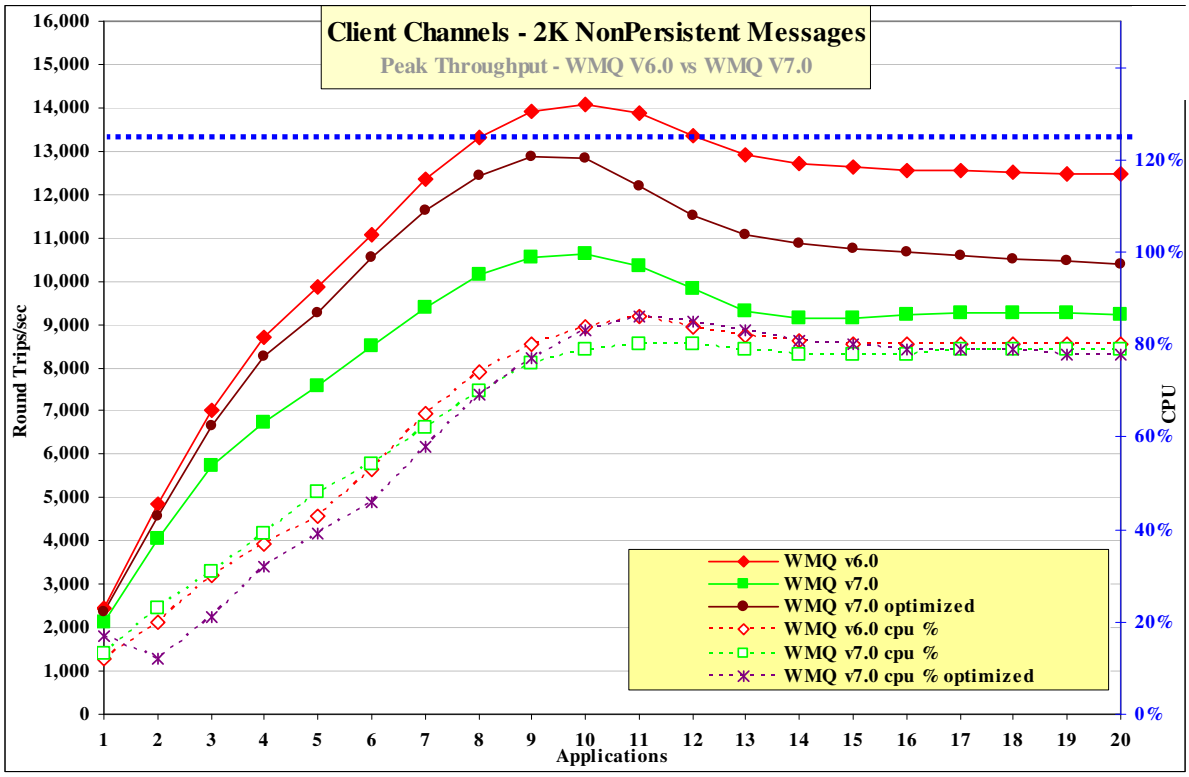


Figure 6 – Performance headline, nonpersistent messages, client channels

Figure 6 and **Table 3** show that the throughput of nonpersistent messages has reduced by 10% with optimised setup and by 24% with default setup when comparing Version 6 to Version 7.

Test name: **clnp**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	(9) **10**	(13920) **14090**	(0.0007) **0.0008**	(84%) **86%**
WebSphere MQ V7.0 Optimised	**9** (10)	**12862** (12821)	**0.0008** (0.0008)	**83%** (86%)
WebSphere MQ V7.0	(9) **10**	(10553) **10638**	(0.0010) **0.0011**	(76%) **79%**

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

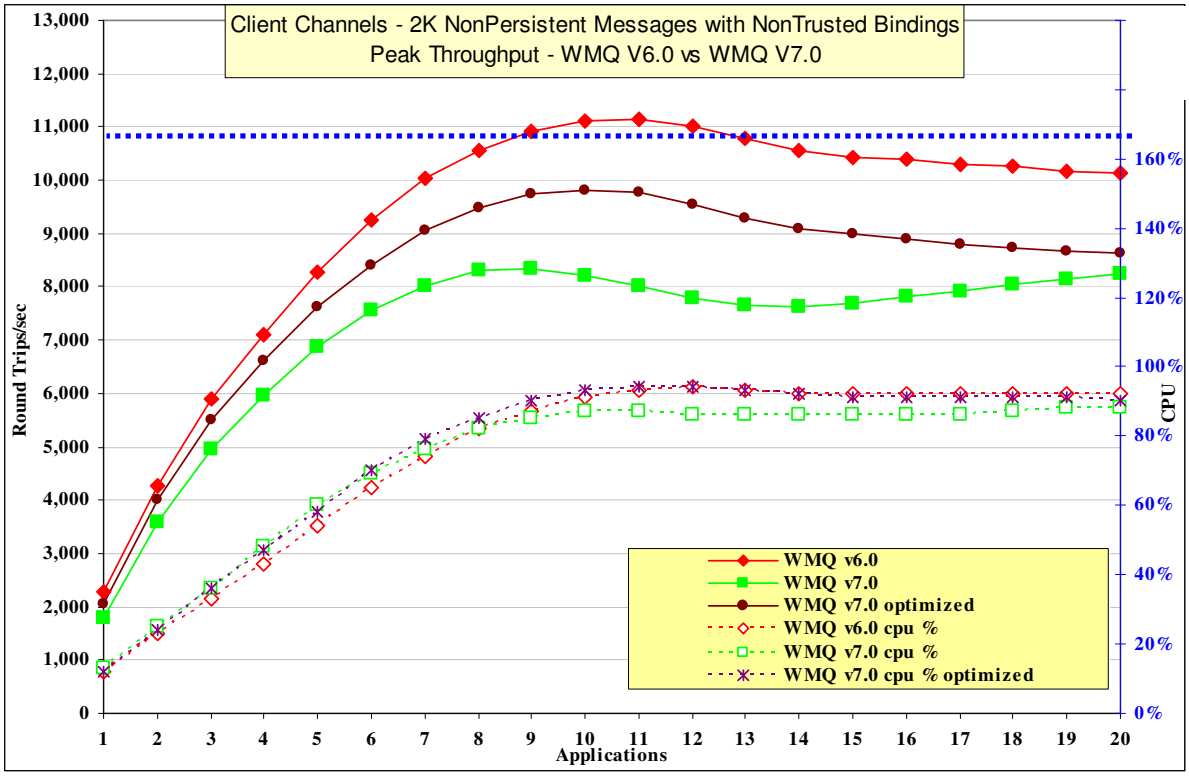


Figure 6a

Test name: **Clnp_nt**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	(9) **11**	(10915) **11129**	(0.0010) **0.0012**	(87%) **93%**
WebSphere MQ V7.0 Optimised	**10** (11)	**9818** (9779)	**0.0012** (0.0013)	**93%** (94%)
WebSphere MQ V7.0	**9** (10)	**8356** (8226)	**0.0013** (0.0014)	**85%** (87%)

The throughput of nonpersistent messages when the channel has used the default MQIBINDTYPE=NORMAL has reduced by 11% with optimised setup and by 22% with default setup when comparing Version 6 to Version 7.

2.2.3 Persistent Messages – Client Channels

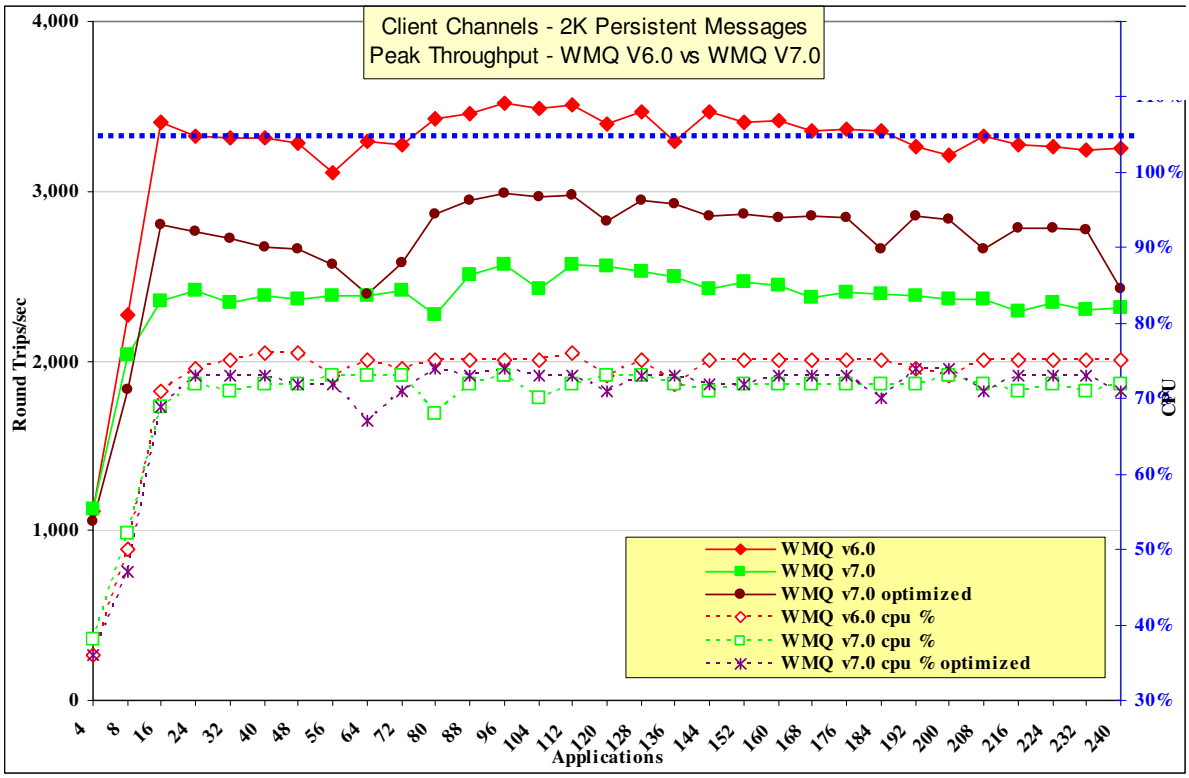


Figure 7 – Performance headline, persistent messages, client channels

Figure 7 and **Table 4** show that the throughput of persistent messages has decreased by 17% when using optimized setup and 29% using default setup when comparing Version 6 to Version 7.

Test name: **Clpm**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**96**	**3520**	**0.0325**	**75%**
WebSphere MQ V7.0 Optimised	**96**	**2986**	**0.0386**	**74%**
WebSphere MQ V7.0	**96**	**2565**	**0.0441**	**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

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. The Client channel definition was optimised by using sharecnv=0.

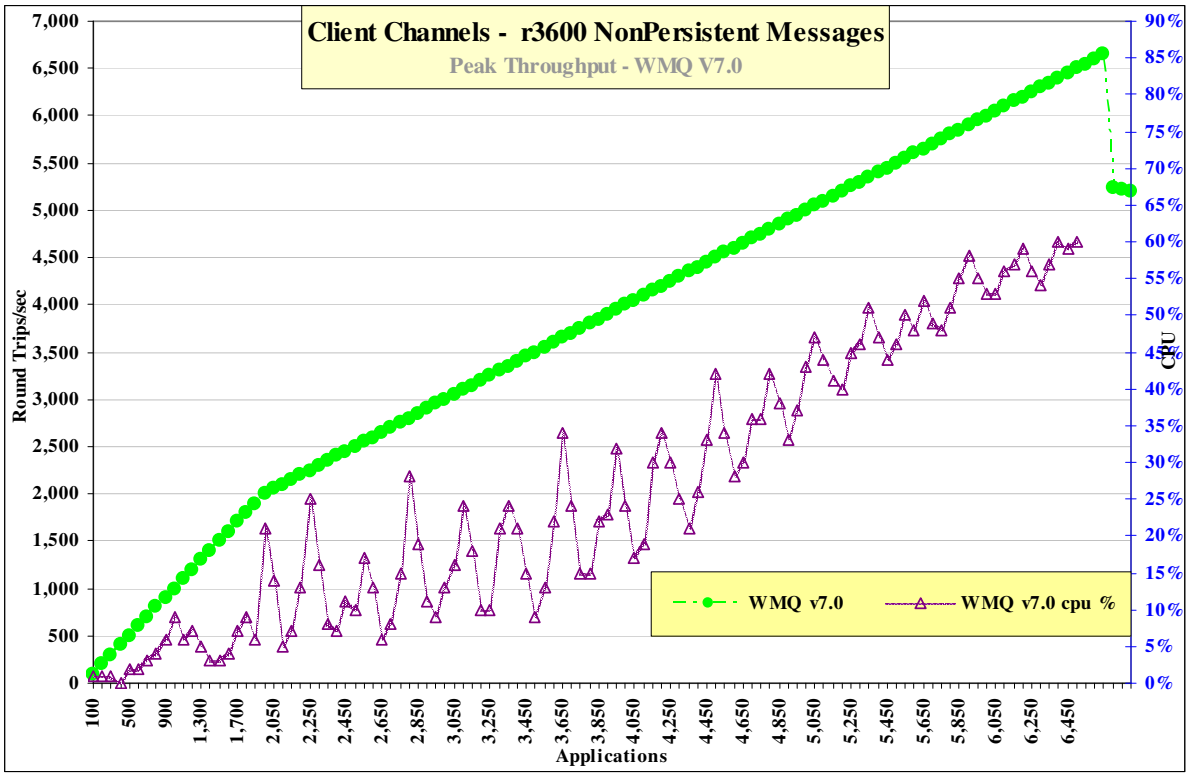


Figure 8 – 1 round trip per driving application per second, client channels, nonpersistent messages

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

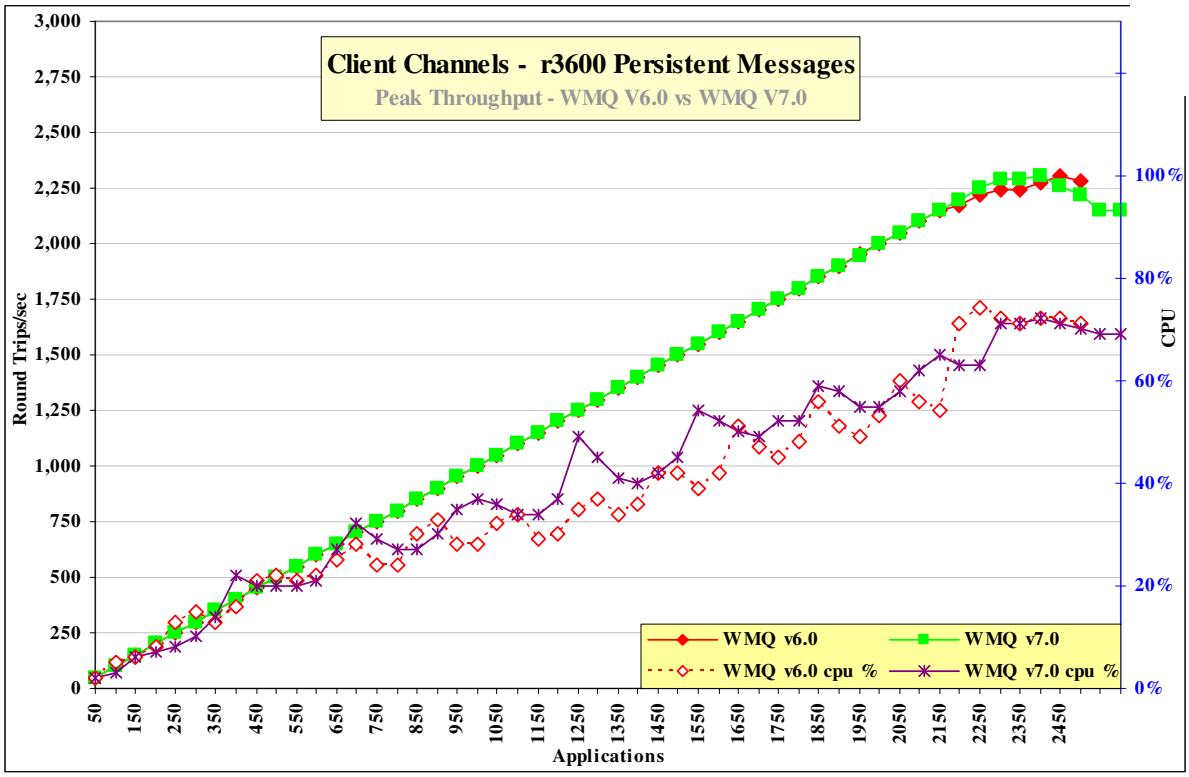


Figure 9 – 1 round trip per driving application per second, client channels, persistent messages

Figure 8, Figure 9 and **Table 5** shows that WebSphere MQ V7.0 has similar throughput and cpu characteristics.

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
clnp_r3600	**6600**	3,600	**6588**	**0.0070**	**61%**
clpm_r3600	**(2150)**		**(2149)**	**(0.0059)**	**(65%)**
	2250	3,600	**2249**	**0.0070**	**63%**
WMQ v6.0	2150		2149	0.0138	54%

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 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 7.

2.3 Distributed Queuing Test Scenario

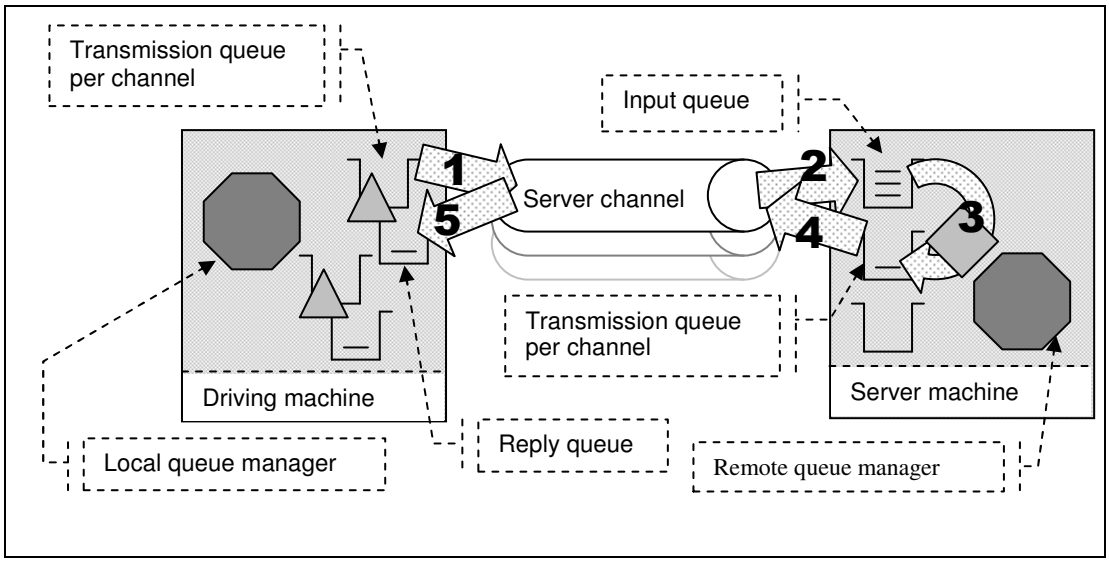


Figure 10 – Server channels between two queue managers

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 = STANDARD'.

2.3.1 Nonpersistent Messages – Server Channels

Figure 11 , **Figure 11a**, and **Figure 12** show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario (see **Figure 10** on the previous page), and WebSphere MQ V6.0 compared to Version 7.

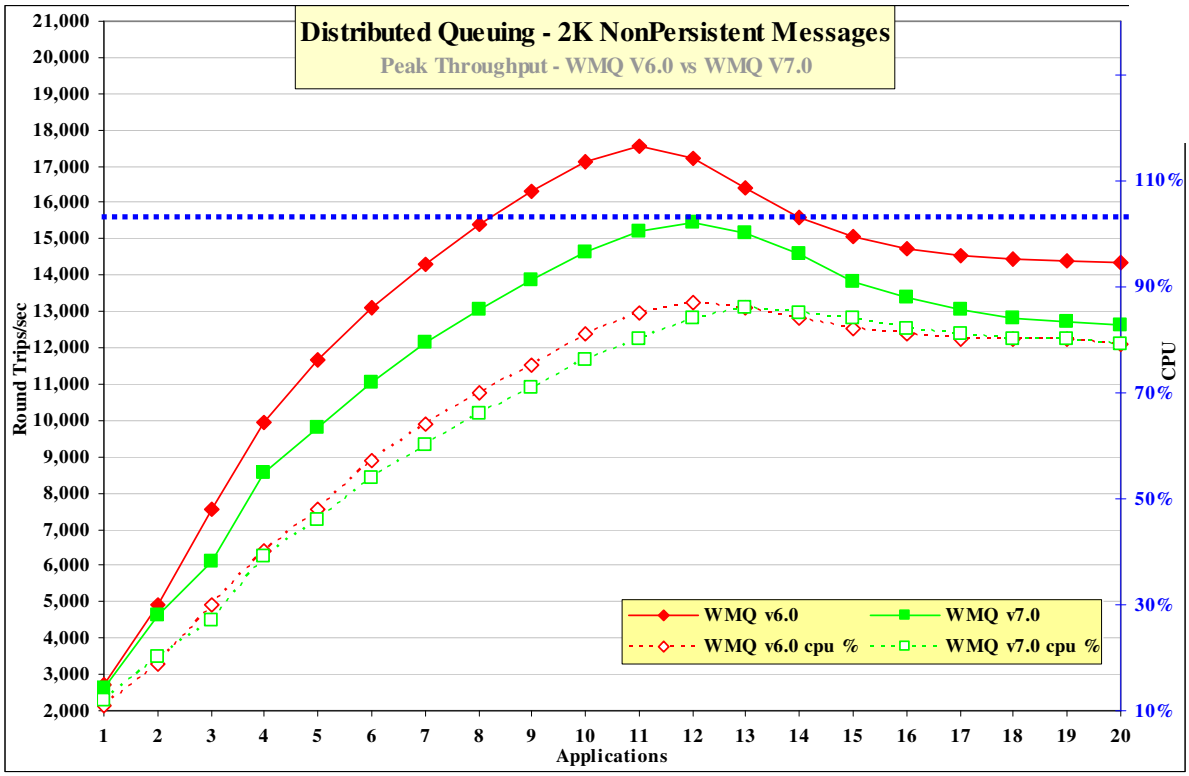


Figure 11 – Performance headline, nonpersistent messages, server channels

Figure 11 and **Table 6** show that the throughput of nonpersistent messages has reduced by 12% comparing Version 6 to Version 7.

Test name: **dqnp**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**11** (12)	**17560** (17220)	**0.0008** (0.0008)	**85%** (87%)
WebSphere MQ V7.0	(11) **12**	(15193) **15425**	(0.0009) **0.0009**	(80%) **84%**

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

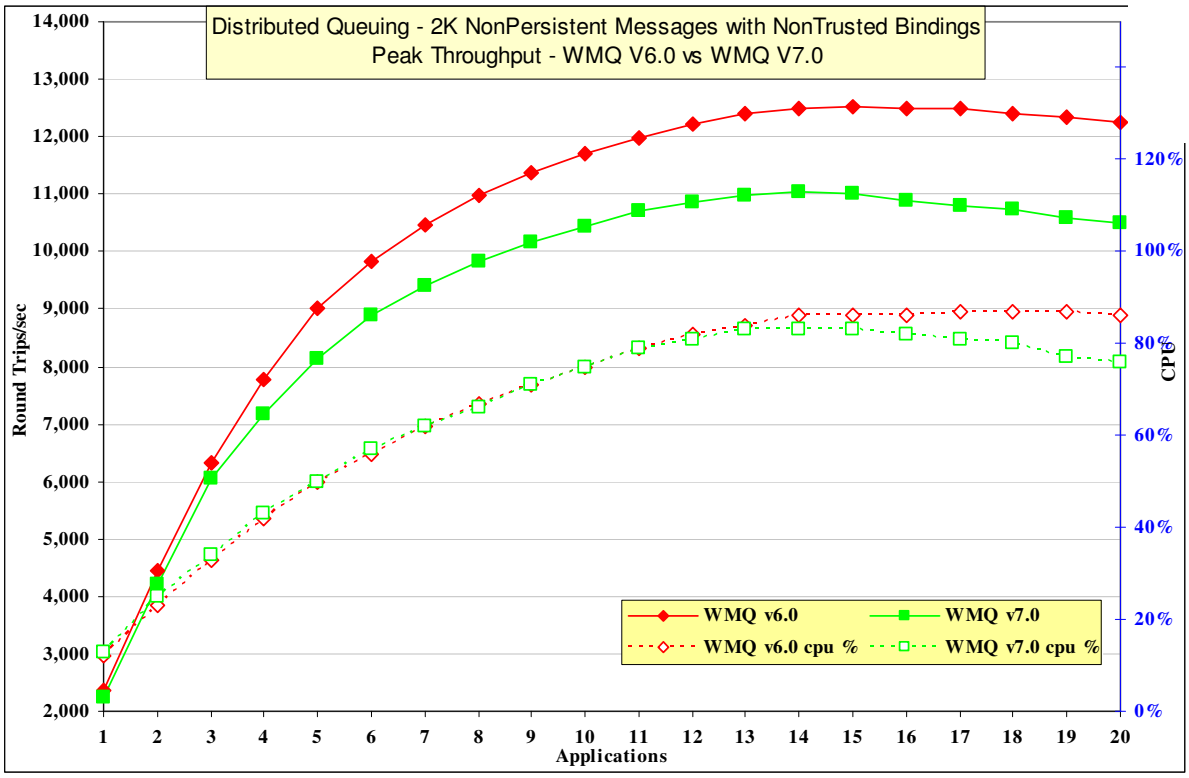


Figure 11a

Test name: **Dqnp_nt**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**15**	**12533**	**0.0014**	**86%**
WebSphere MQ V7.0	**15**	**11041**	**0.0015**	**83%**

The throughput of nonpersistent messages has reduced by 10% comparing Version 6 to Version 7.

2.3.3 Persistent Messages – Server Channels

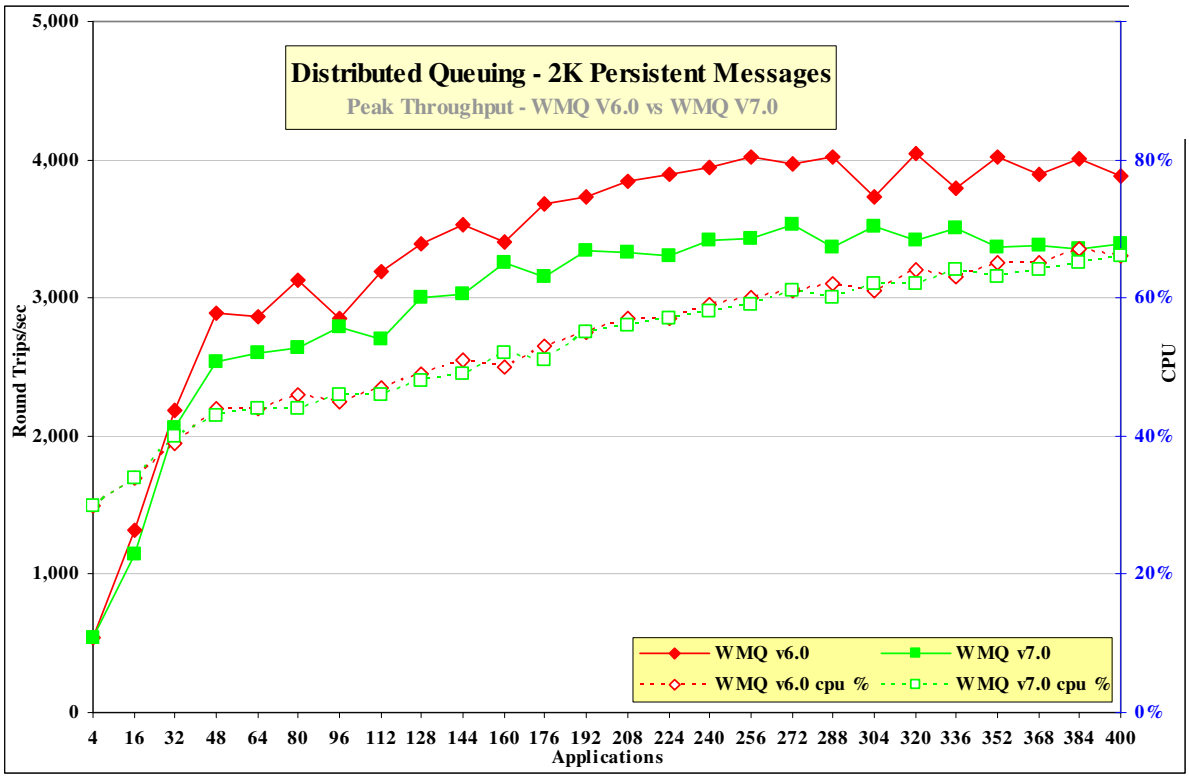


Figure 12 – Performance headline, persistent messages, server channels

Figure 12 and **Table 7** show that the throughput of persistent messages has declined by 12% comparing Version 6 to Version 7.

Test name: **dqpm**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**320**	**4049**	**0.0892**	**64%**
WebSphere MQ V7.0	**272**	**3424**	**0.1030**	**61%**

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.

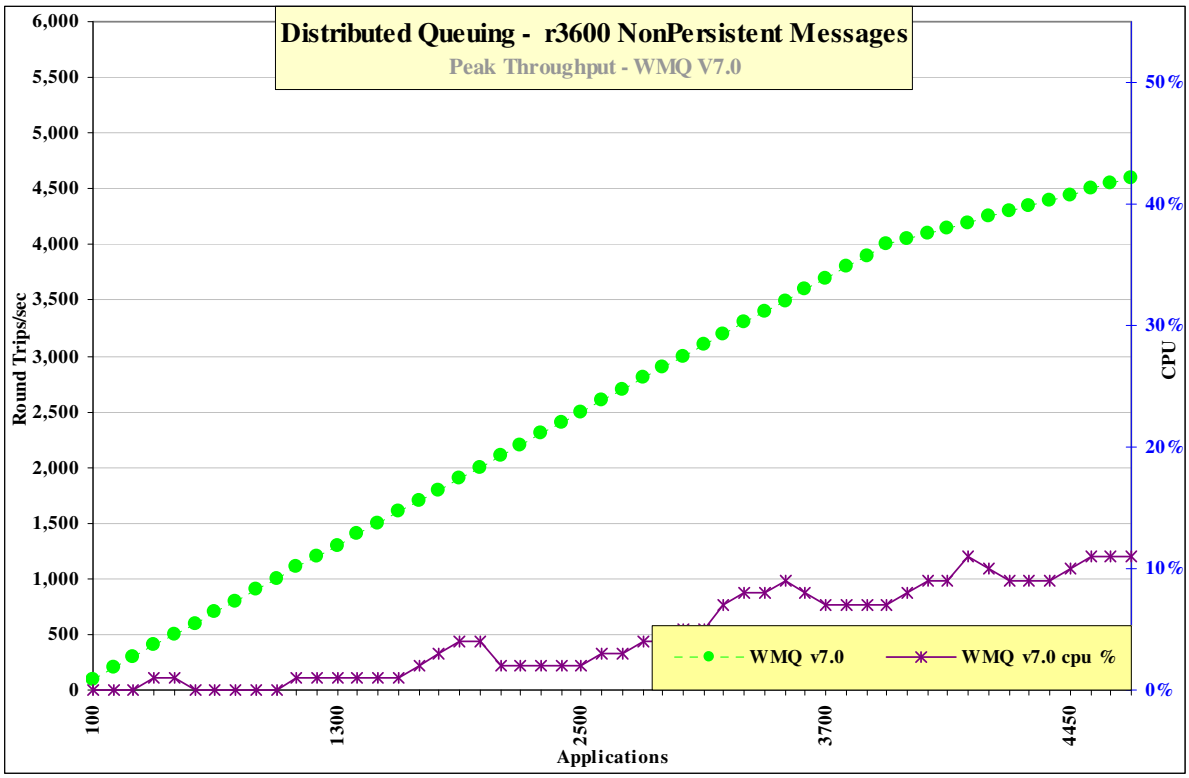


Figure 13 – 1 round trip per driving application per second, server channel, nonpersistent messages

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

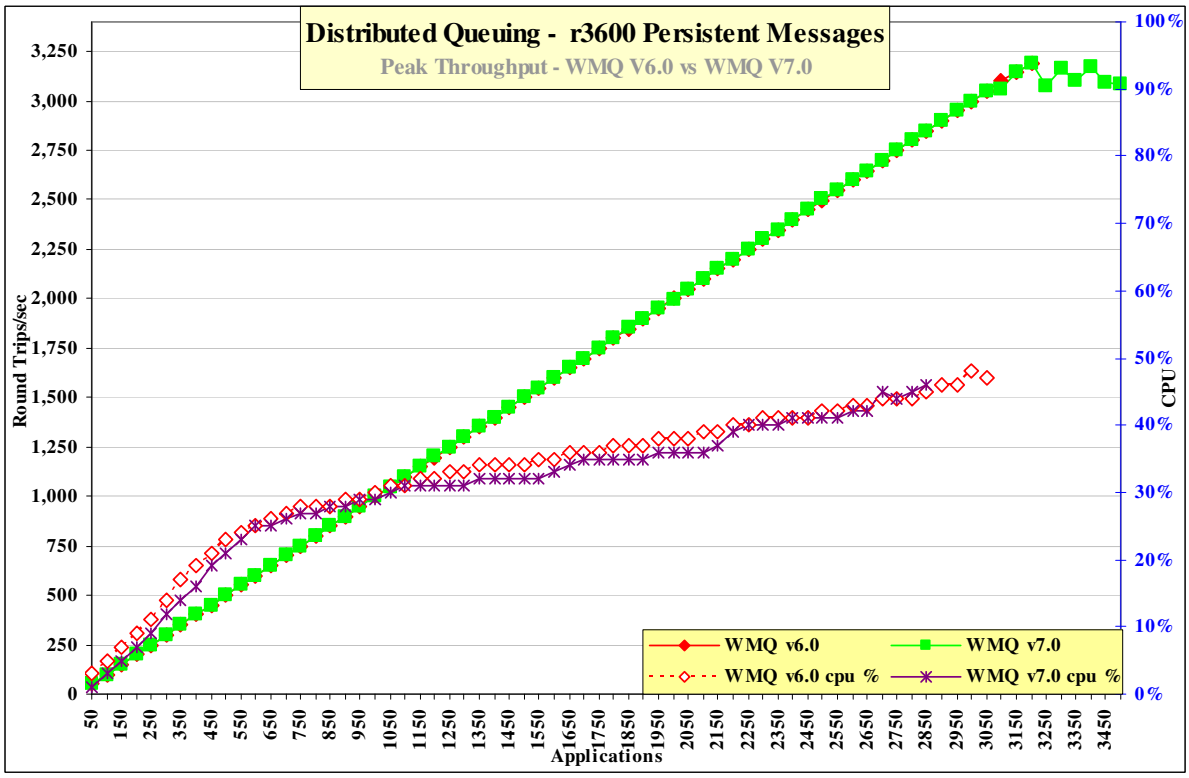


Figure 14 – 1 round trip per driving application per second, server channel, persistent messages

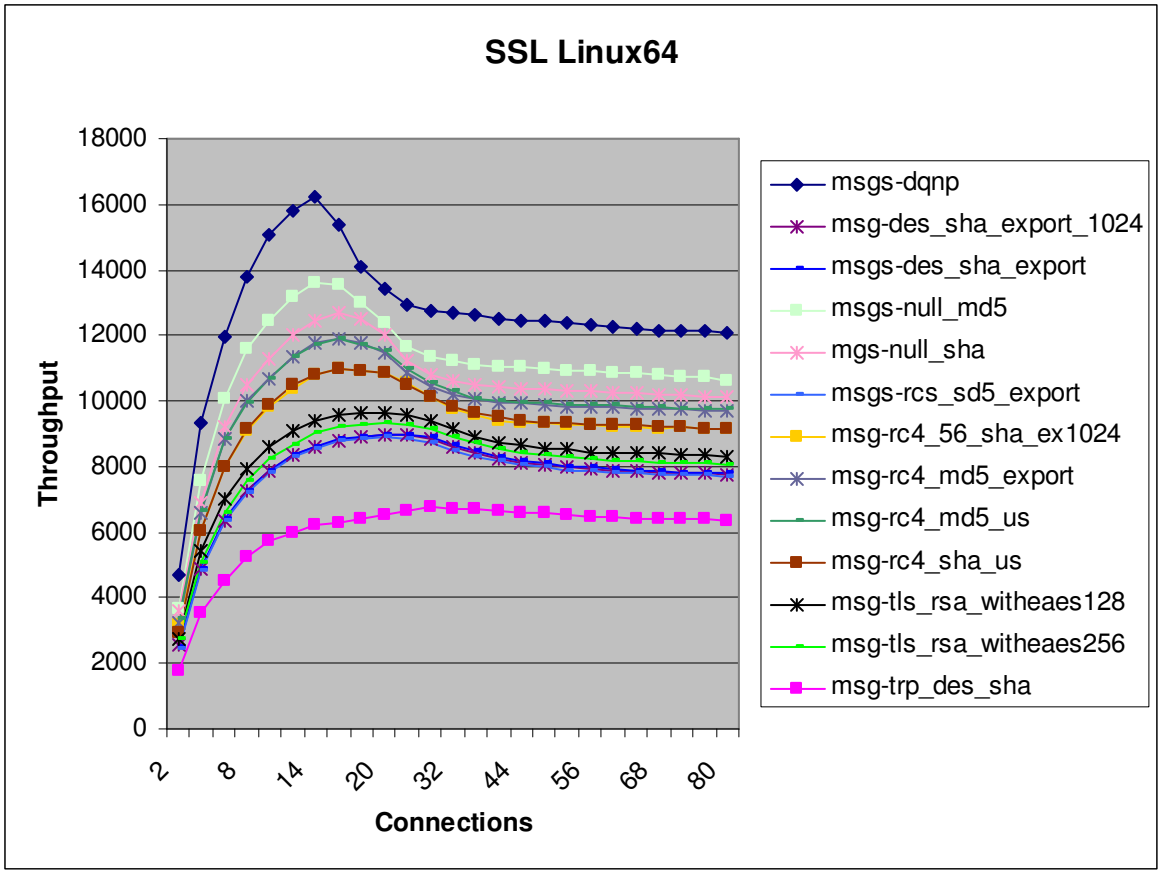
Figure 13, **Figure 14** and **Table 8** shows how WebSphere MQ V7.0 is similar in performance as version 6.

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
dqnp_r3600	**4600**	3,600	**4598**	**0.0008**	**30%**
dqpm_r3600 (WebSphereMQ v6.0)	**3050** (3050)	3,600	**3049** (3048)	**0.187** (0.041)	**50%** (48%)

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.

2.3.5 SSL



There was no SSL used in the dqnp measurement. The use of cipher specifications null_sha and null_md5 (which do not encrypt the messages) degrade throughput by 13% and 18% respectively. The other 10 cipher specs degrade throughput by up to 50%. The following table shows the throughput, response time and server cpu% with 40 connections.

cipher spec	throughput	Response time	Server cpu%
DQNP (no SSL)	12525	0.003 seconds	79%
triple_des_sha_us	6672	0.007 seconds	86%
tls_rsa_with_aes_256_cbc_sha	8534	0.005 seconds	66%
rc4_sha_us	9508	0.004 seconds	80%
rc4_md5_us	10013	0.004 seconds	79%
rc4_md5_export	9956	0.004 seconds	79%
rc4_56_sha_export1024	9423	0.004 seconds	79%
rc2_md5_export	8169	0.006 seconds	83%
null_sha	10413	0.004 seconds	79%
null_md5	11073	0.004 seconds	79%

des_sha_export	8314	0.005 seconds	82%
des_sha_export1024	8253	0.005 seconds	82%
tls_rsa_with_aes_128_cbc_sha	8577	0.005 seconds	82%

The GSKIT level is 7.0.4.20

3 Large Messages

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

3.1.1 50bytes to 32Kb

Figure 15 show that the response time for MQPut/MQGet for np message sizes between 50bytes and 32Kb.

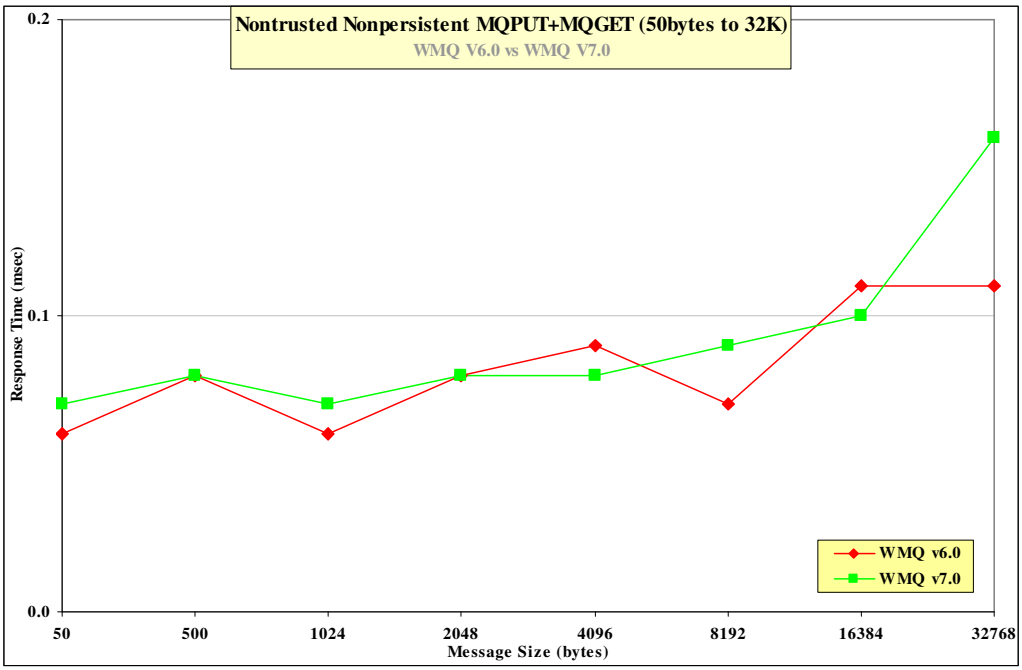


Figure 15 –The effect of nonpersistent message size on MQI response time (50byte - 32K)

Figure 16 show that the response for MQPut/MQGet pairs for pers message sizes between 50bytes and 16Kb.

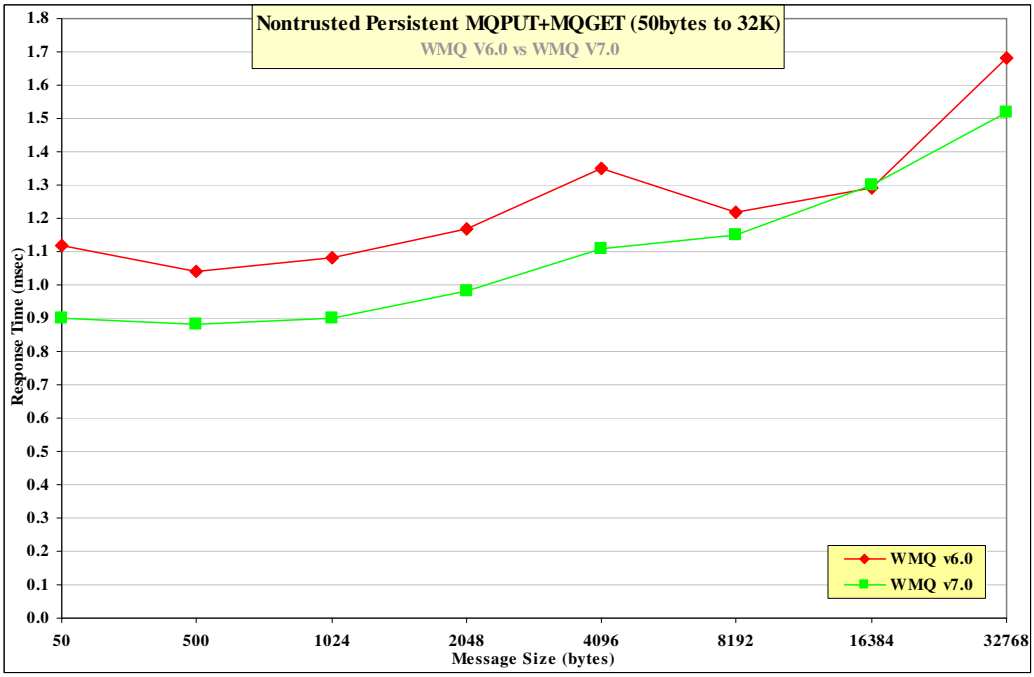


Figure 16 –The effect of persistent message size on MQI response time (50byte - 32K)

3.1.2 32Kb to 2Mb

Figure 17 show that the response time for MQPut/MQGet pairs has improved for all nonpersistent message sizes between 32Kb and 2Mb.

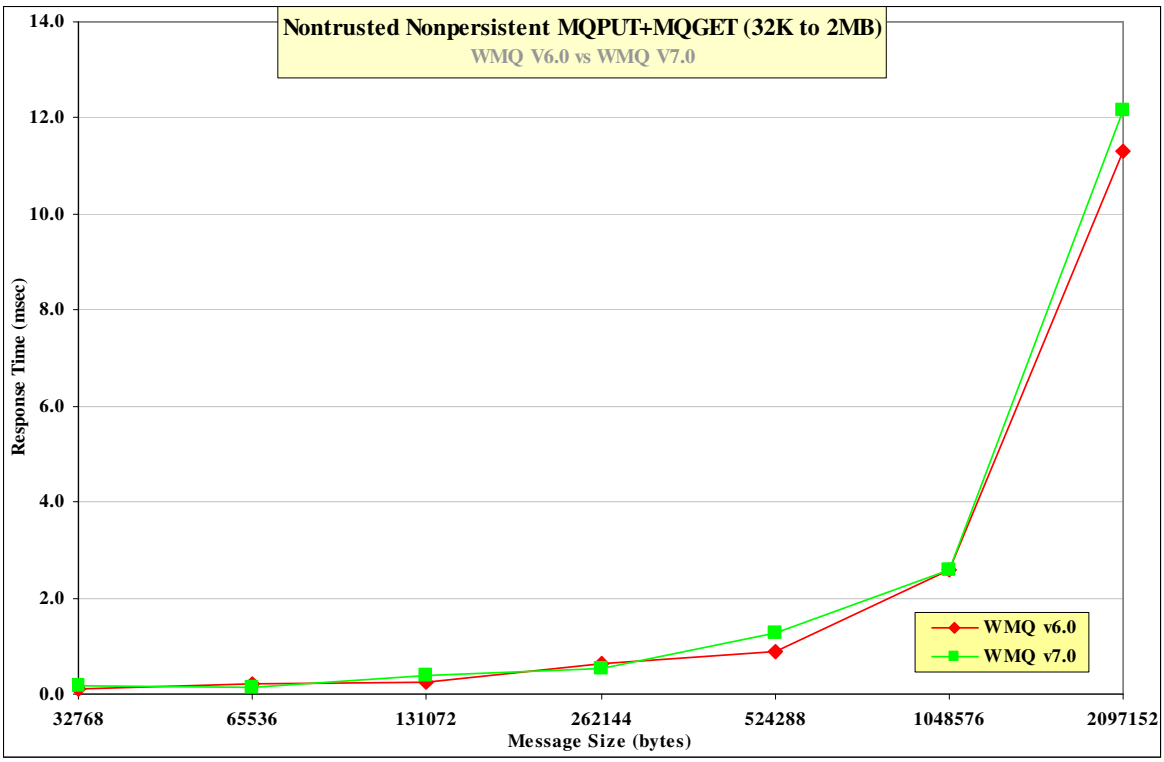


Figure 17 –The effect of nonpersistent message size on MQI response time (32K – 2Mb)

Figure 18 show that the response for MQPut/MQGet pairs for persistent message sizes between 32Kb and 2Mb.

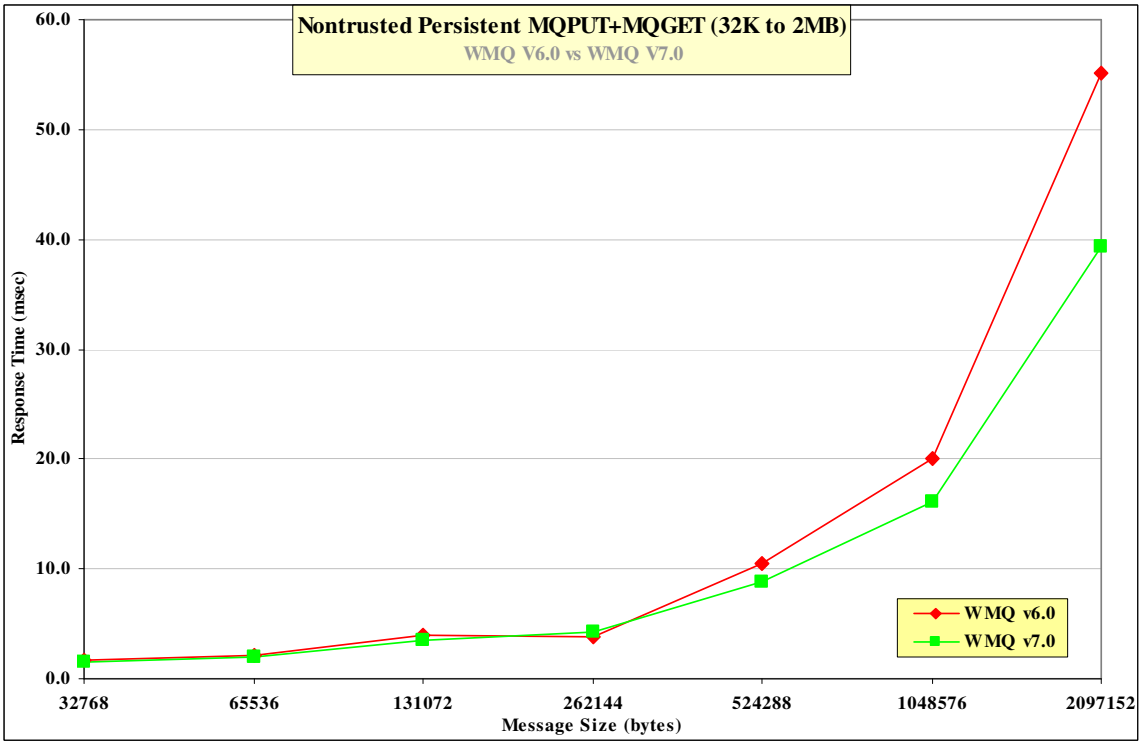


Figure 18 –The effect of persistent message size on MQI response time (32K – 2Mb)

3.1.3 2Mb to 100Mb

Figure 19 Response time for MQPut/MQGet pairs for NP message between 2Mb and 100Mb.

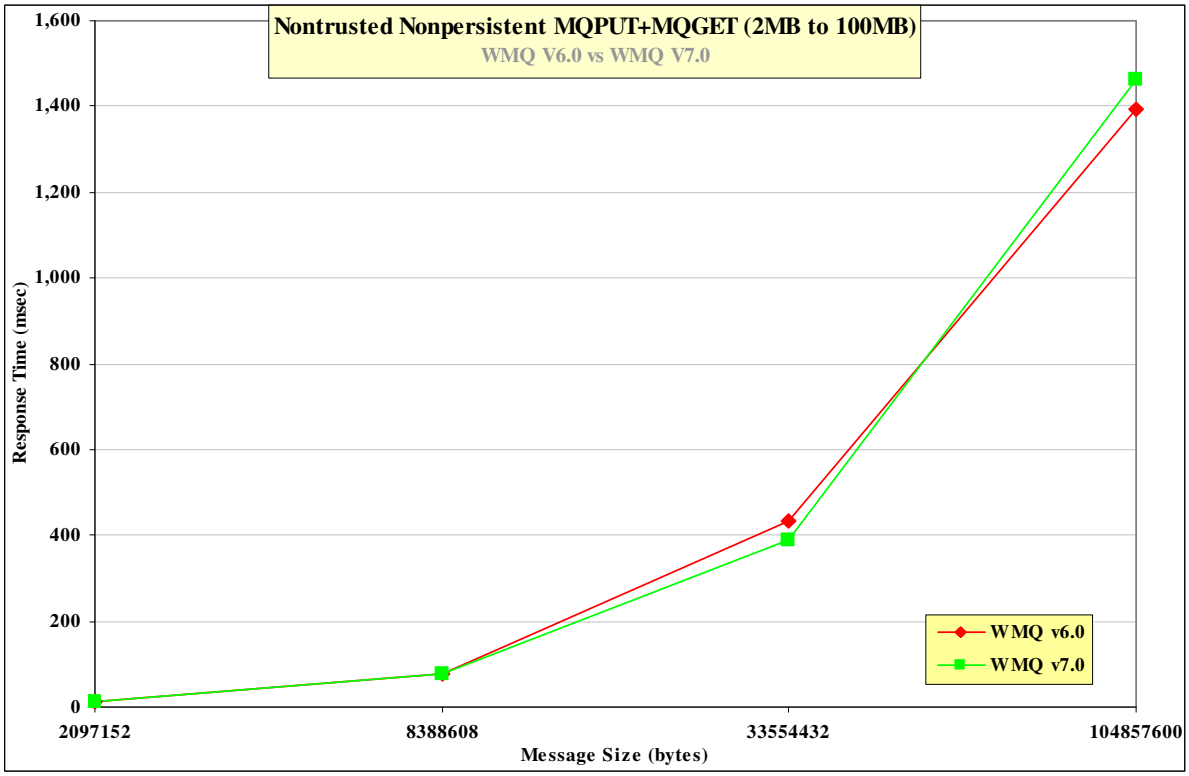


Figure 19 –The effect of nonpersistent message size on MQI response time (2Mb – 100Mb)

Figure 20 The response for MQPut/MQGet pairs for persistent message sizes between 2Mb and 100Mb.

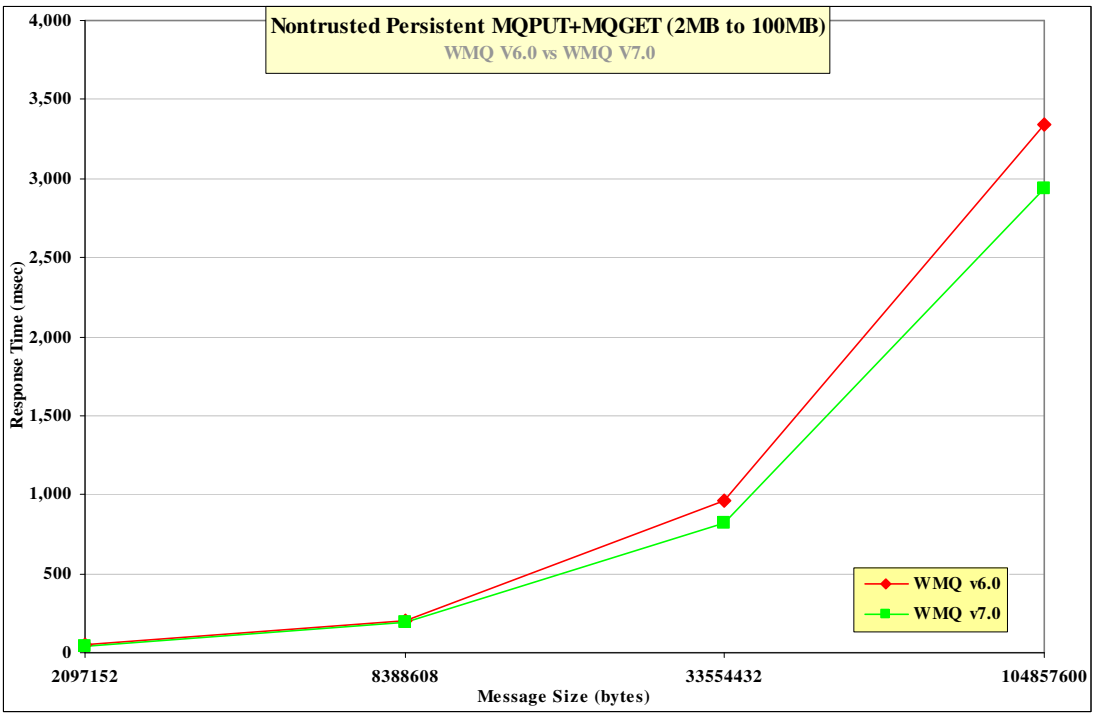


Figure 20 –The effect of persistent message size on MQI response time (2Mb – 100Mb)

3.2 20K Messages

3.2.1 Local Queue Manager

Figure 21 and **Figure 22** show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

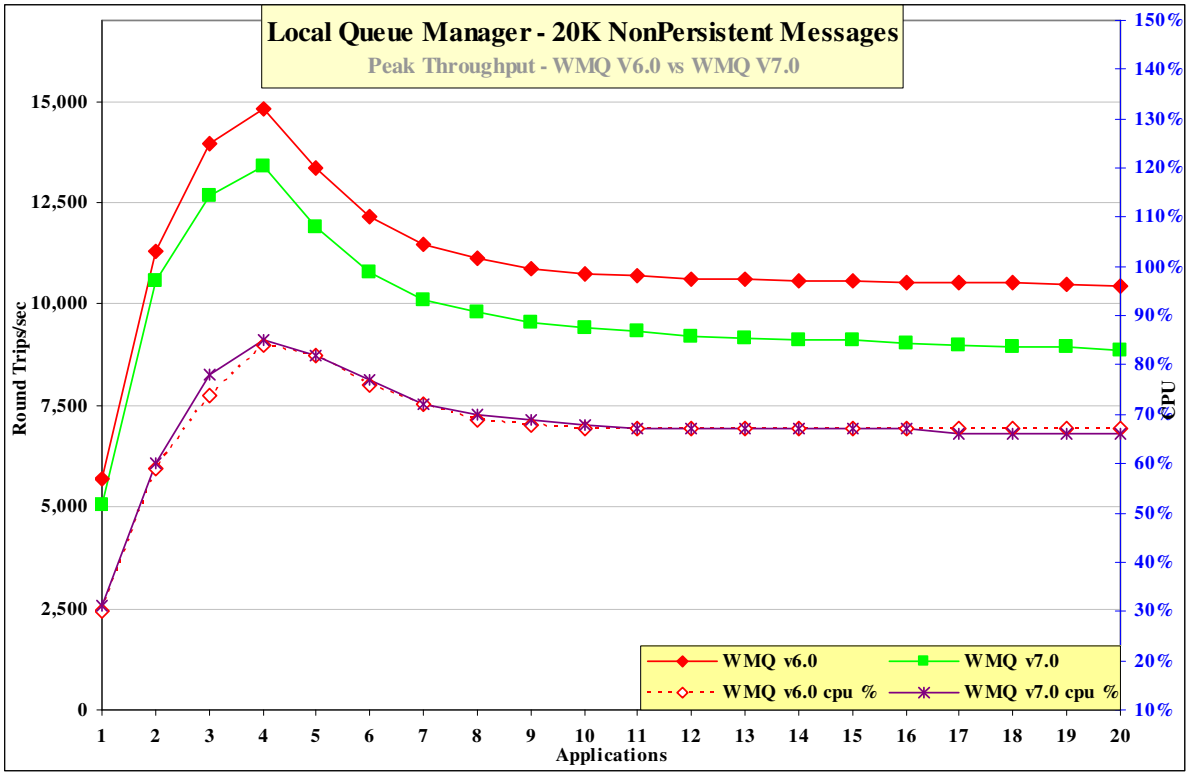


Figure 21 – 20K nonpersistent messages, local queue manager

Figure 21 and **Table 9** show that the throughput of nonpersistent messages has degraded by 12% comparing Version 7 to Version 6.

Test name: **local_np_20K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**4**	**14830**	**0.0003**	**84%**
WebSphere MQ V7.0	**4**	**13419**	**0.0004**	**85%**

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.*

3.2.1.1 Persistent Messages

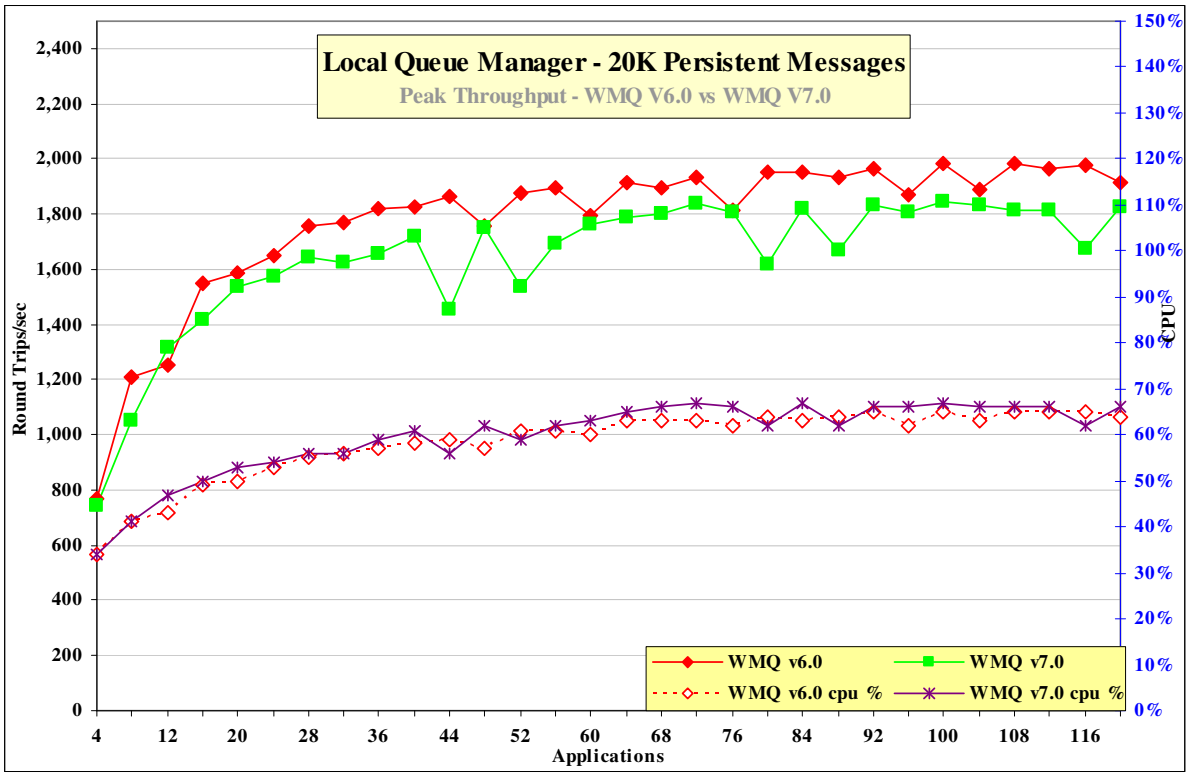


Figure 22 – 20K persistent messages, local queue manager

Figure 22 and **Table 10** show that the throughput of persistent messages has degraded by 7% comparing Version 6 to Version 7.

Test name: **local_pm_20K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**108**	**1986**	**0.0265**	**65%**
WebSphere MQ V7.0	**100**	**1844**	**0.065**	**67%**

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. .

3.2.2 Client Channel

Fig 23 and **Fig24** show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

3.2.2.1 Nonpersistent Messages

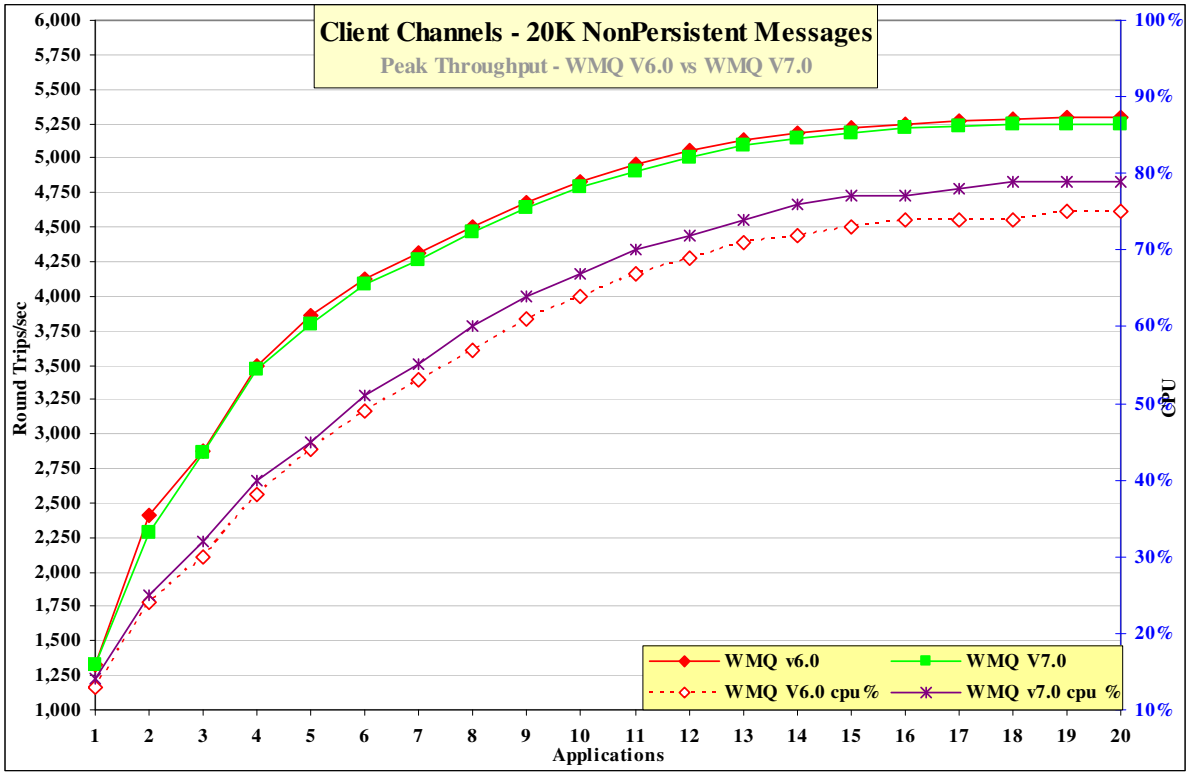


Figure 23 – 20K nonpersistent messages, client channels

Fig (hyperlink)and **Table 11** show that the throughput of nonpersistent messages is similar when comparing Version 6 to Version 7.

Test name: **clnp_20K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**38**	**5296**	**0.04**	**75%**
WebSphere MQ V7.0	**38**	**5249**	**0.04**	**79%**

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

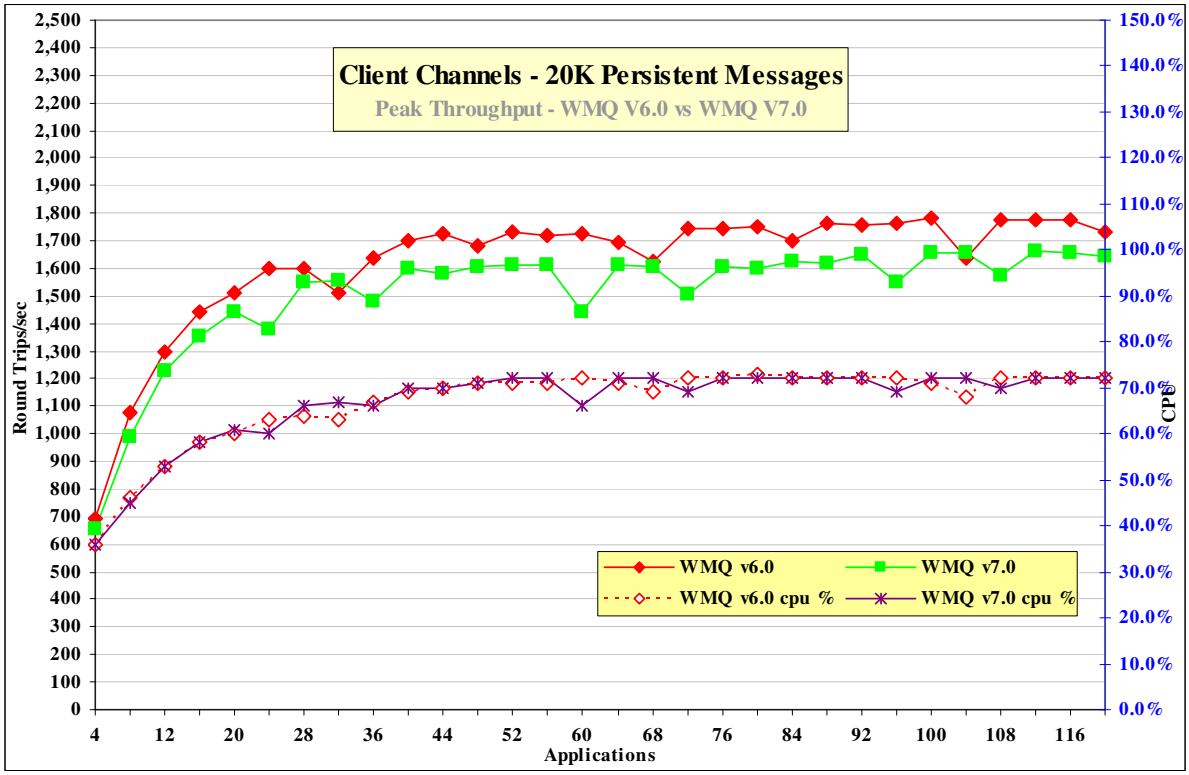


Figure 24 – 20K persistent messages, client channels

Fig 24 and **Table 12** show that the throughput of persistent messages has degraded by 6% comparing Version 6 to Version 7.

Test name: **clpm_20K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	(112) **100**	(1774) **1784**	(0.074) **0.074**	(72%) **71%**
WebSphere MQ V7.0	**112** (100)	**1665** (1664)	**0.077** (0.071)	**45%** (72%)

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 25 and **fig 26** 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

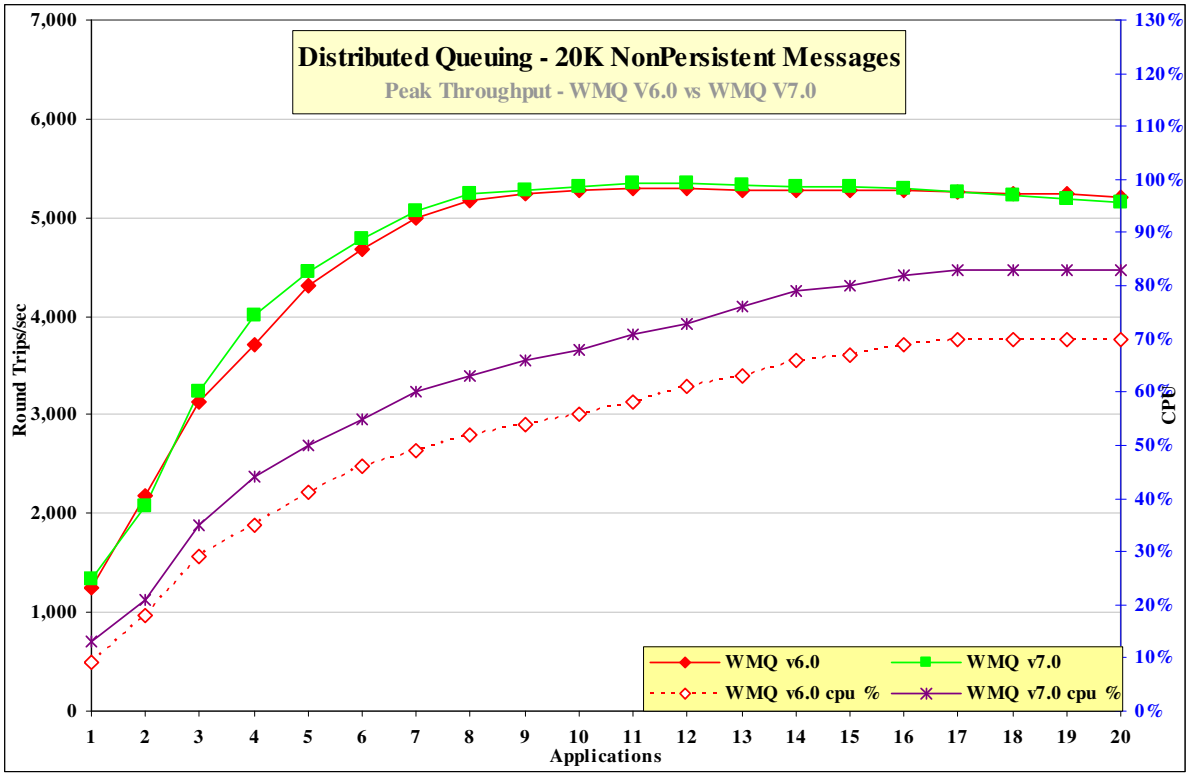


Figure 25 – 20K nonpersistent messages, distributed queuing

Figure 25 and **Table 13** show that the throughput of nonpersistent messages is similar when comparing Version 6 to Version 7.

Test name: **dqnp_20K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**12**	**5301**	**0.002**	**61%**
WebSphere MQ V7.0	**12**	**5348**	**0.0027**	**73%**

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.

3.2.3.2 Persistent Messages

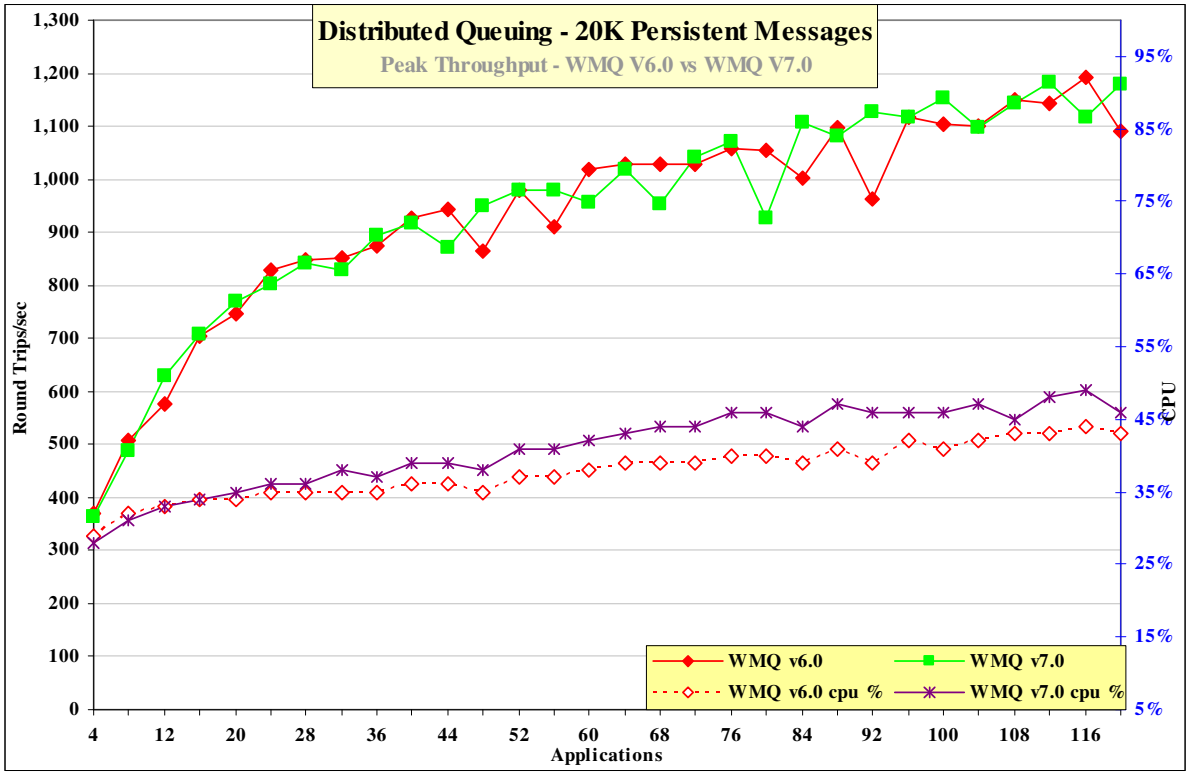


Figure 26 – 20K persistent messages, distributed queuing

Fig 26 and **Table 14** show that the throughput of nonpersistent messages is similar when comparing Version 6 to Version 7.

Test name: **dqpm_20K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**116**	**1193**	**0.111**	**43%**
WebSphere MQ V7.0	**112** (116)	**1182** (1117)	**0.114** (0.1247)	**48%** (49%)

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

Fig 27 and **fig 28** 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

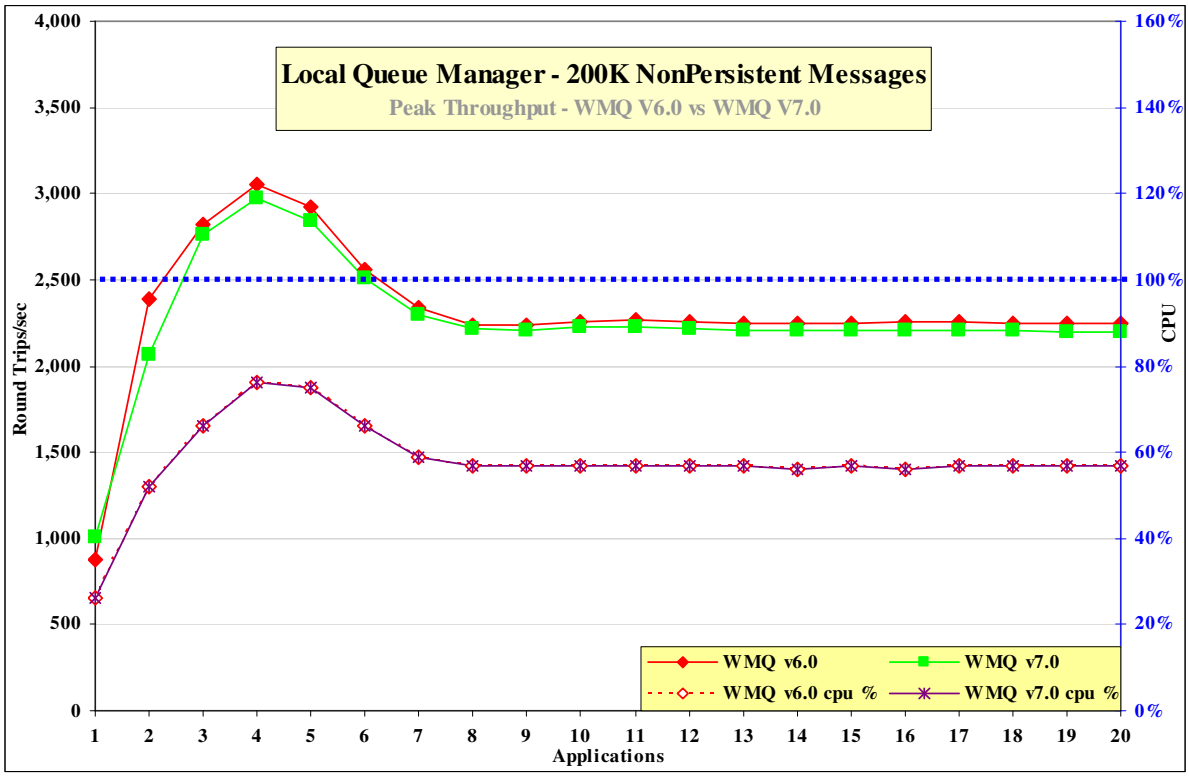


Figure 27 – 200K nonpersistent messages, local queue manager

and **Table 15** show that the throughput of nonpersistent messages is similar comparing Version 6 to Version 7.

Test name: **local_np_200K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**4**	**3048**	**0.002**	**76%**
WebSphere MQ V7.0	**4**	**2968**	**0.002**	**76%**

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

3.3.1.2 Persistent Messages

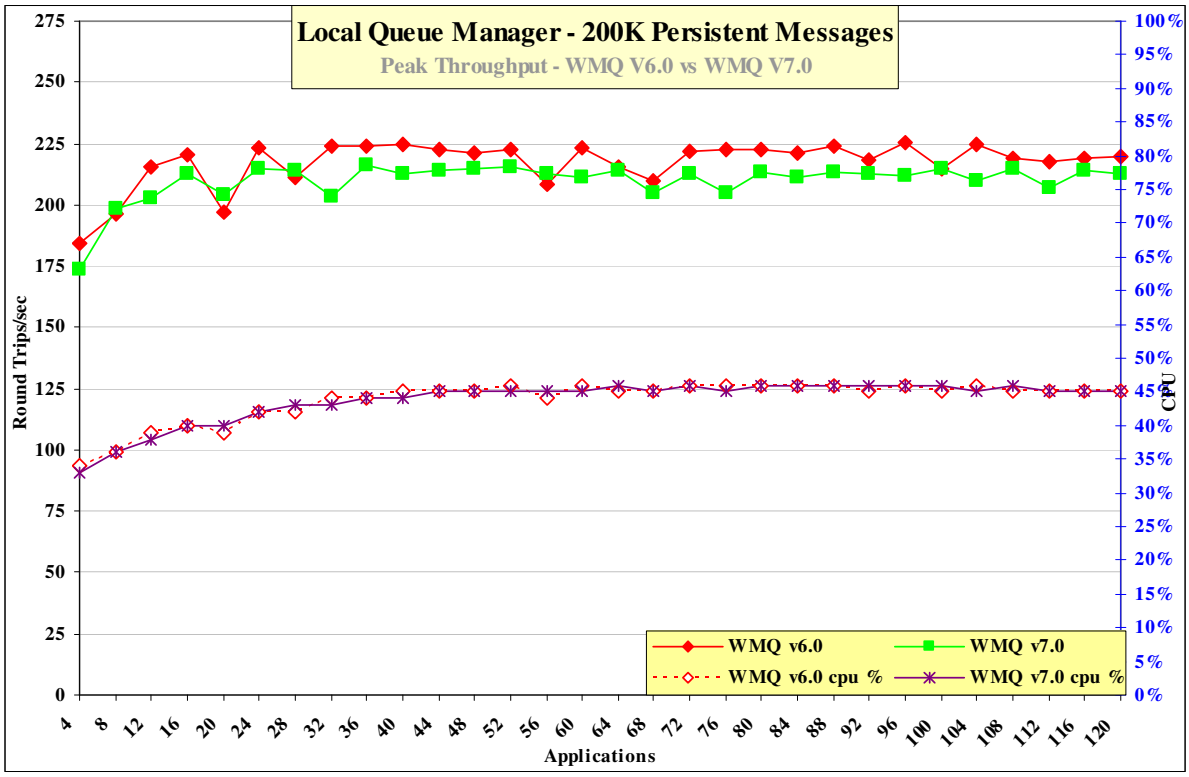


Figure 28 – 200K persistent messages, local queue manager

Fig and **Table 16** show that the throughput of persistent messages has degraded by 3% comparing Version 6 to Version 7.

Test name: **local_pm_200K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**96**	**225**	**0.491**	**46%**
WebSphere MQ V7.0	**36** (96)	**217** (212)	**0.186** (0.640)	**44%** (46%)

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

Fig 29 and **Fig 30** show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

3.3.2.1 Nonpersistent Messages

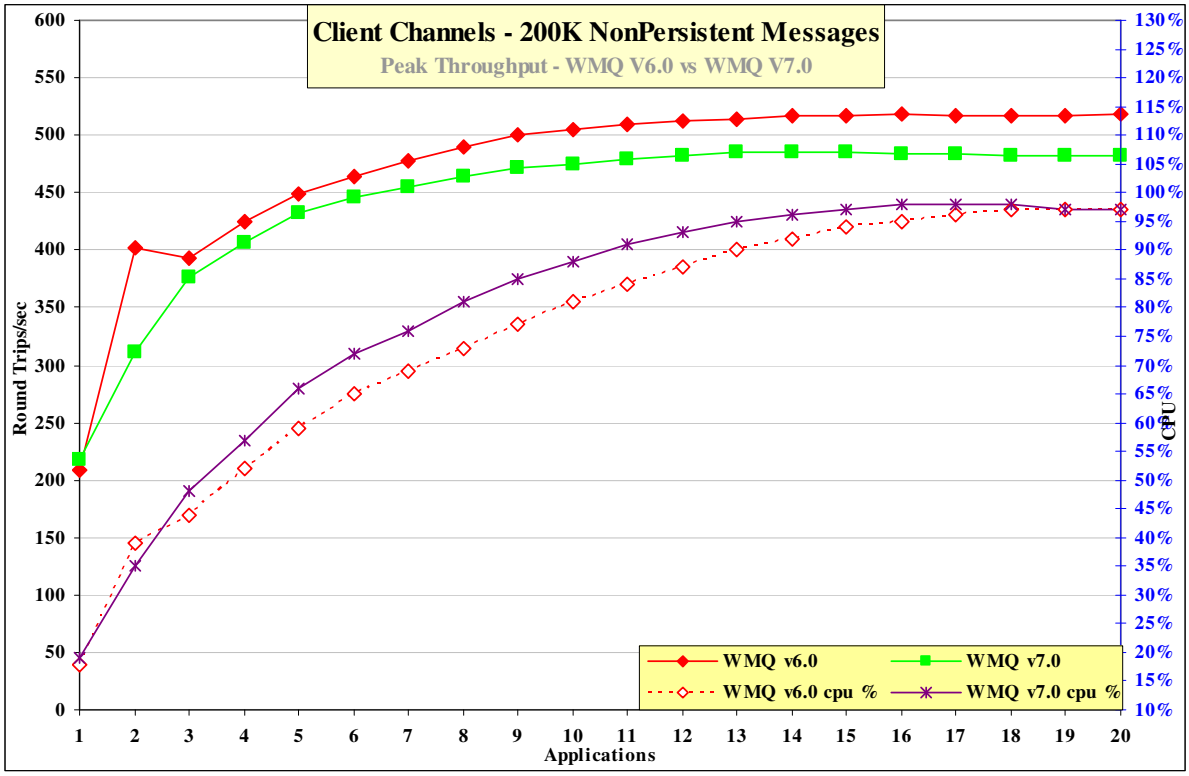


Figure 29 – 200K nonpersistent messages, client channels

and **Table 17** show that the throughput of nonpersistent messages has degraded by 6% when comparing Version 6 to Version 7.

Test name: **clnp_200K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**20**	**518**	**0.046**	**97%**
WebSphere MQ V7.0	**14** (20)	**485** (482)	**0.037** (0.049)	**97%** (97%)

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

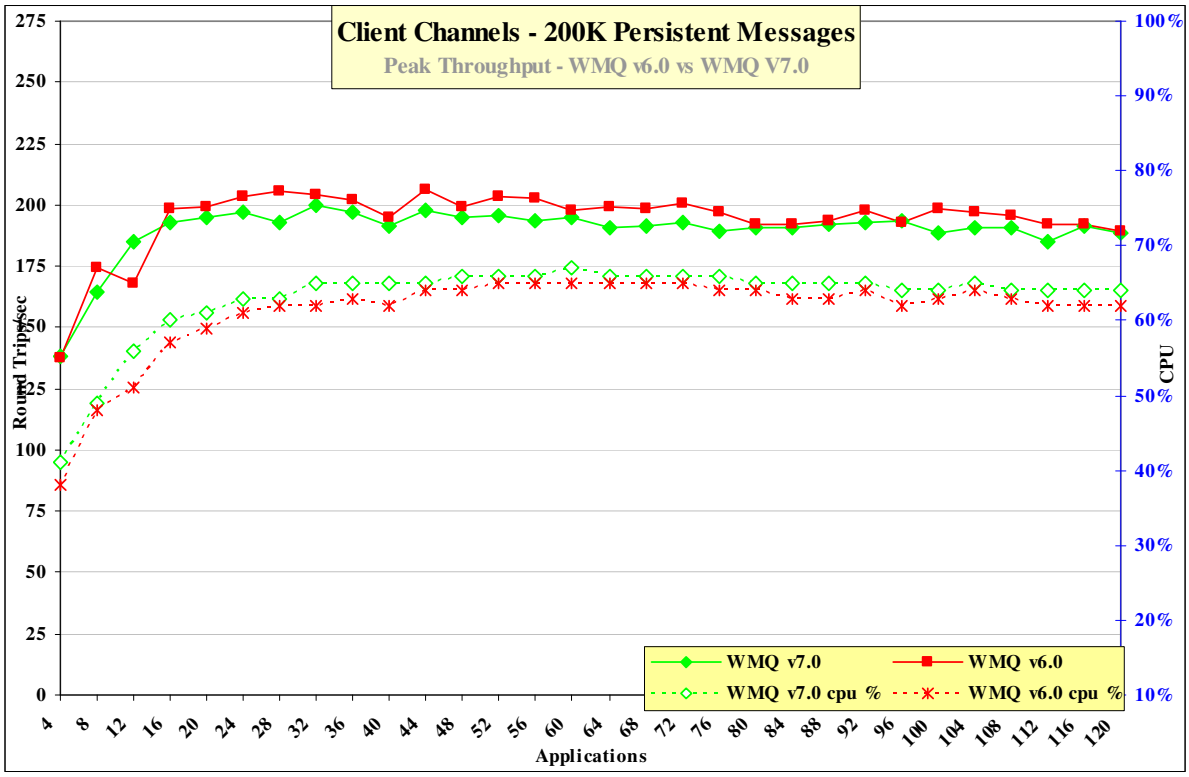


Figure 30 – 200K persistent messages, client channels

Fig 30 and **Table 18** show t that the throughput of persistent messages has degraded by 2% when comparing Version 6 to Version 7.

Test name: **clpm_200K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**44**	**206**	**0.25**	**64%**
WebSphere MQ V7.0	**32**	**200**	**0.192**	**65%**

Table 18 – 200K persistent messages, client channels

3.3.3 Distributed Queuing

Fig 31 and **fig 32** 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

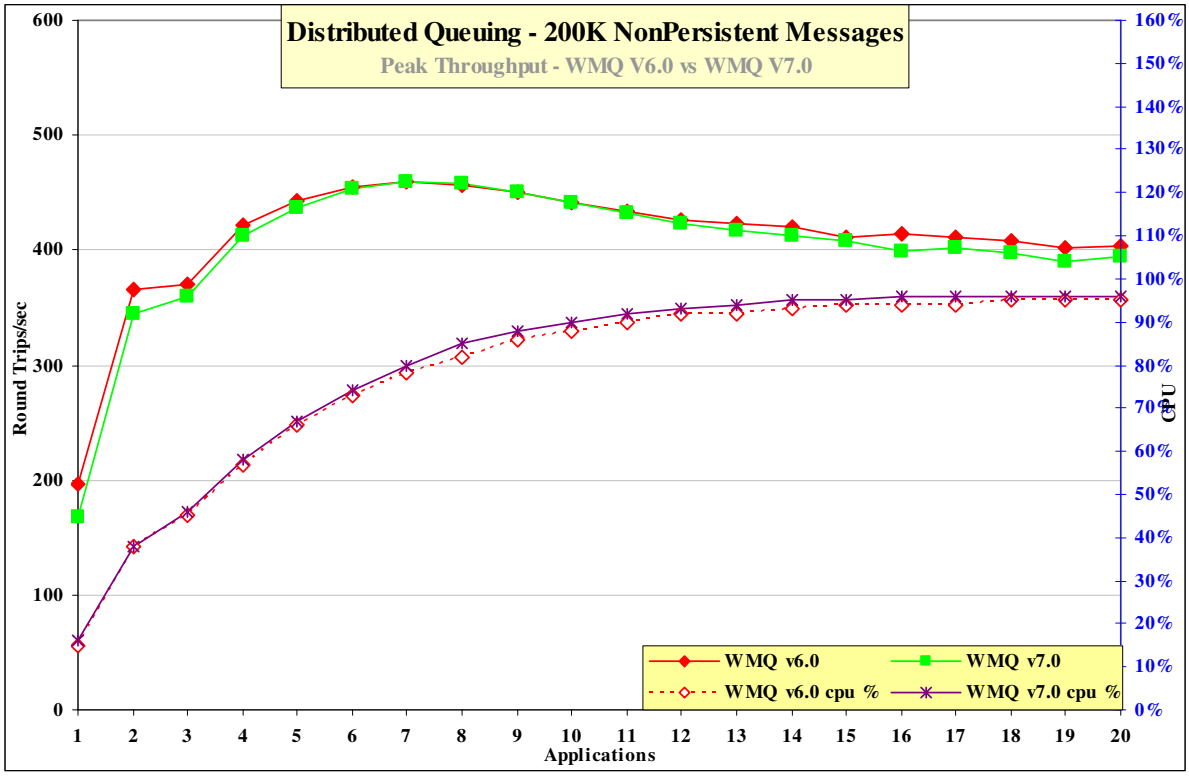


Figure 31 – 200K nonpersistent messages, distributed queuing

Fig 31 and **Table 19** show that the throughput of nonpersistent messages has degraded by 2% when comparing Version 6 to Version 7.

Test name: **dqnp_200K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**7**	**459**	**0.017**	**78%**
WebSphere MQ V7.0	**7**	**459**	**0.018**	**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

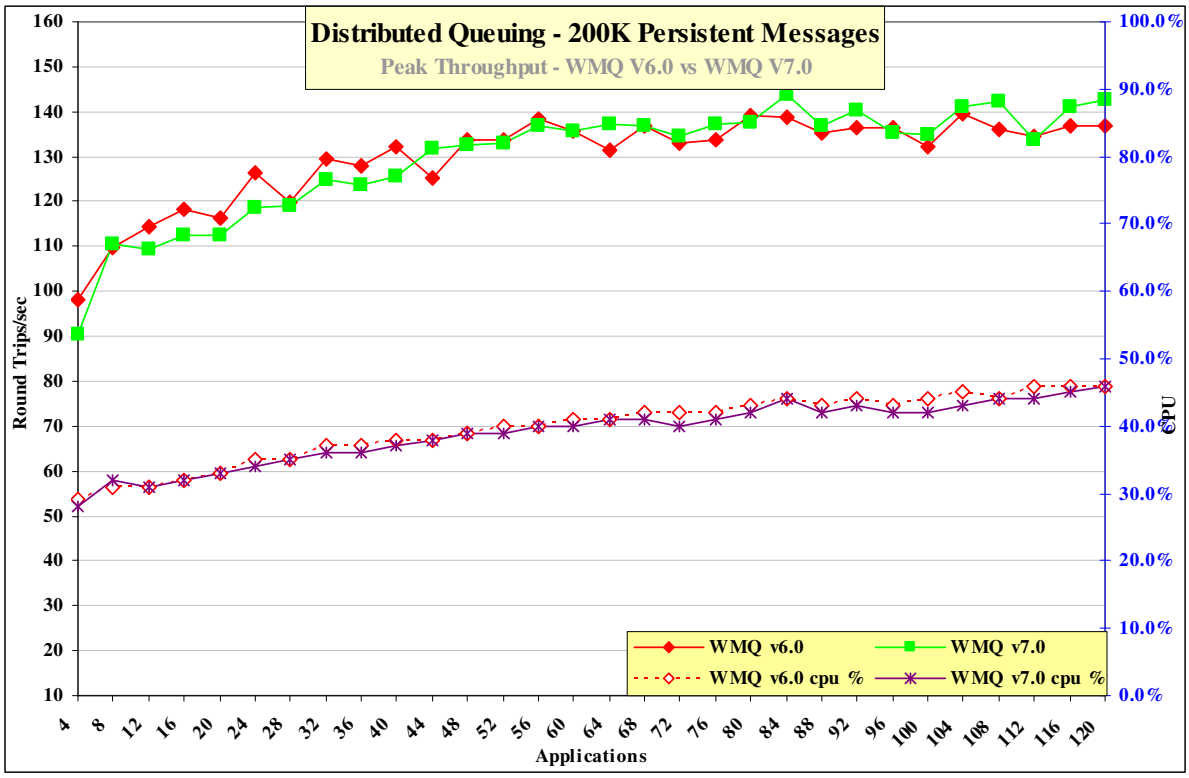


Figure 32 – 200K persistent messages, distributed queuing

Fig 32 and **Table 20** show that the throughput of nonpersistent messages is similar when comparing Version 6 to Version 7.

Test name: **dqpm_200K**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**80** (84)	**139** (139)	**0.70** (0.752)	**43%** (44%)
WebSphere MQ V7.0	**84** (80)	**144** (138)	**0.869** (0.65)	**44%** (42%)

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

Fig 33 and **Fig 34** 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

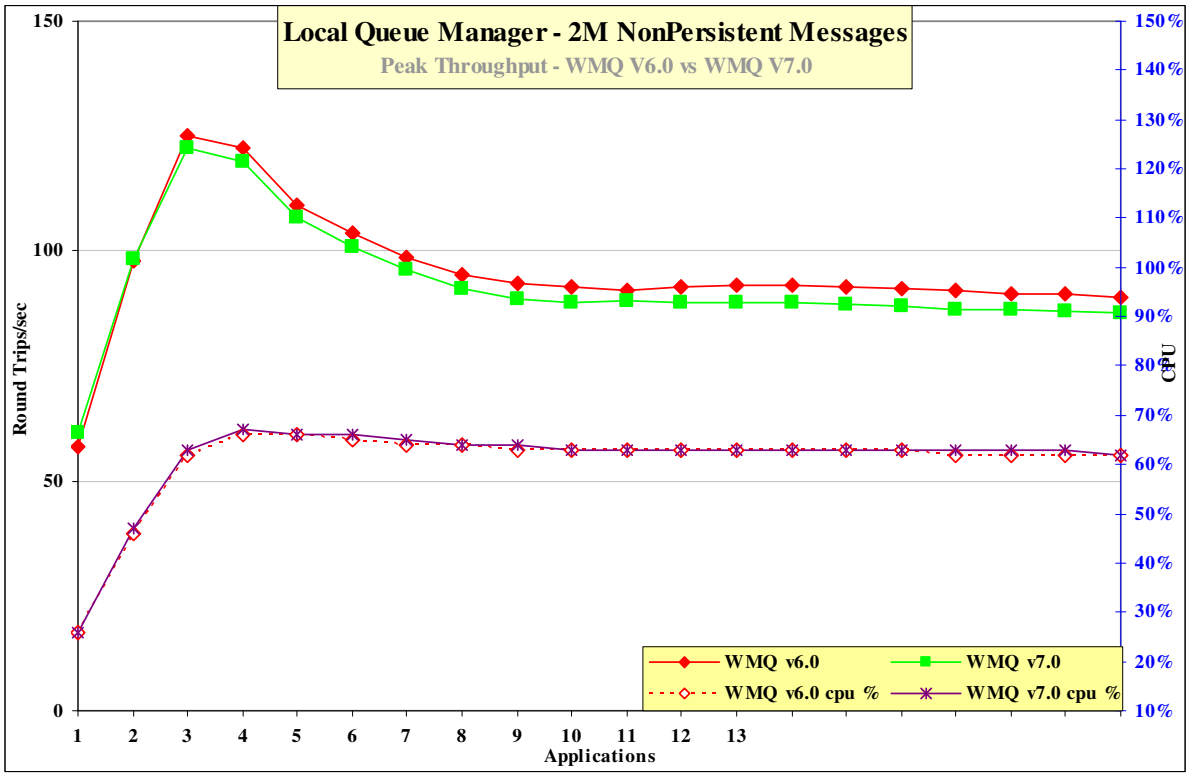


Figure 33 – 2M nonpersistent messages, local queue manager

Fig 33 and **Table 21** show that the throughput of nonpersistent messages has degraded by 2% comparing Version 6 to Version 7.

Test name: **local_np_2M**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**3** (20)	**125** (90)	**0.029** (0.265)	**62%** (62%)
WebSphere MQ V7.0	**3** (20)	**123** (87)	**0.028** (0.273)	**63%** (62%)

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

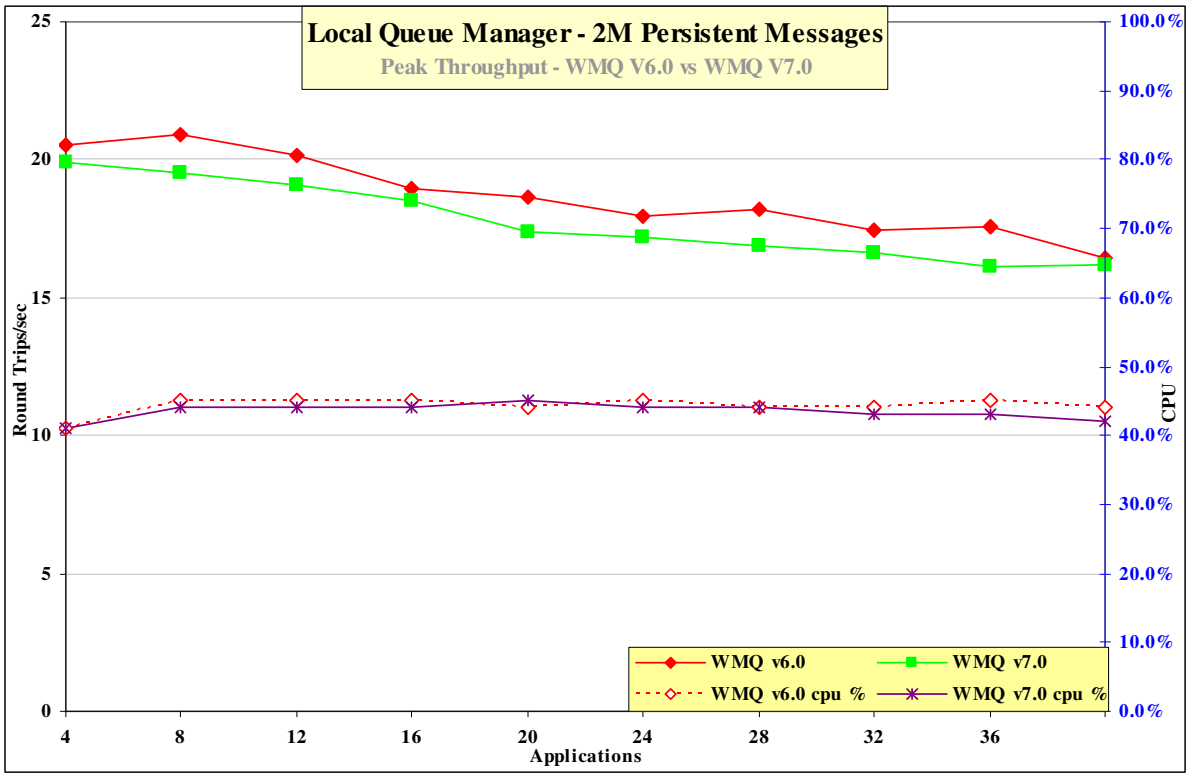


Figure 34 – 2M persistent messages, local queue manager

Fig 34 and **Table 22** show that the throughput of persistent messages has degraded by 5% when comparing Version 6 to Version 7.

Test name: **local_pm_2M**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**8**	**21**	**0.412**	**45%**
WebSphere MQ V7.0	**4** (8)	**20** (19)	**0.212** (0442)	**41%** (44%)

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 35 and **fig 36** show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

3.4.2.1 Nonpersistent Messages

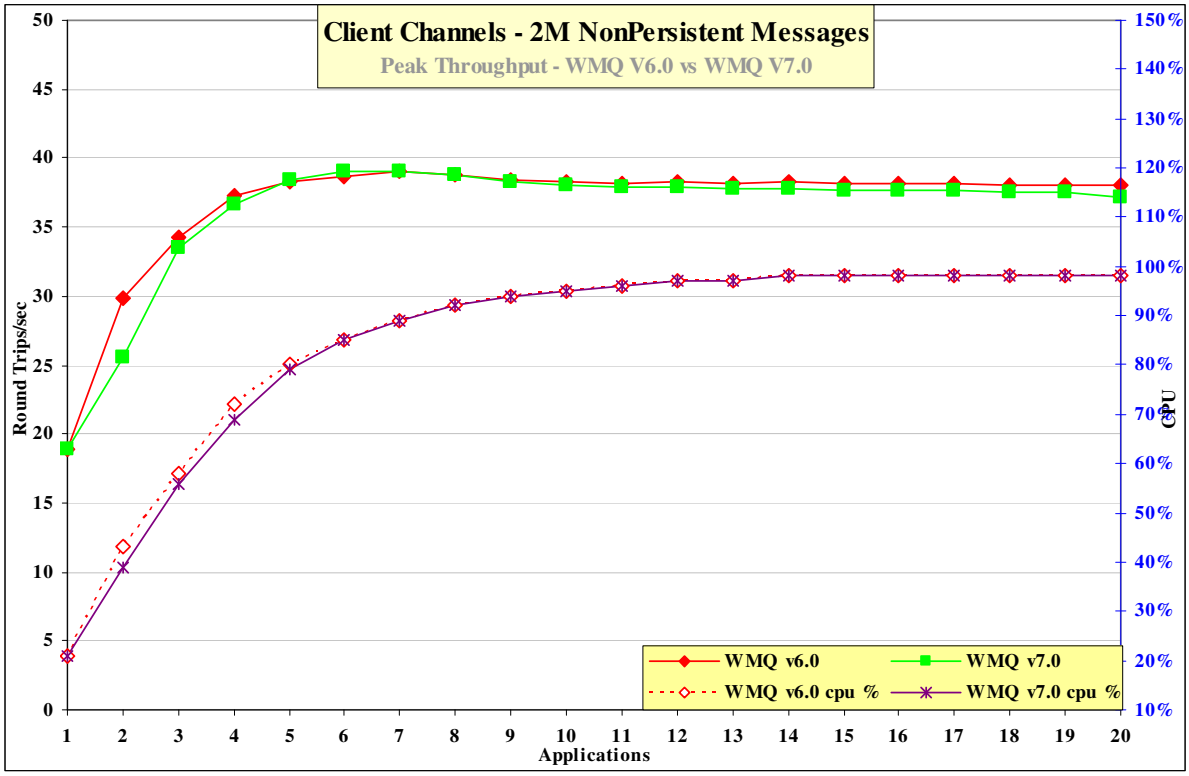


Figure 35 – 2M nonpersistent messages, client channels

Figure 35 and **Table 23** show that the peak throughput of nonpersistent messages is similar when comparing Version 6 to Version 7.

Test name: **clnp_2M**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**7**	**39**	**0.208**	**89%**
WebSphere MQ V7.0	**7**	**39**	**0.209**	**85%**

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.

3.4.2.2 Persistent Messages

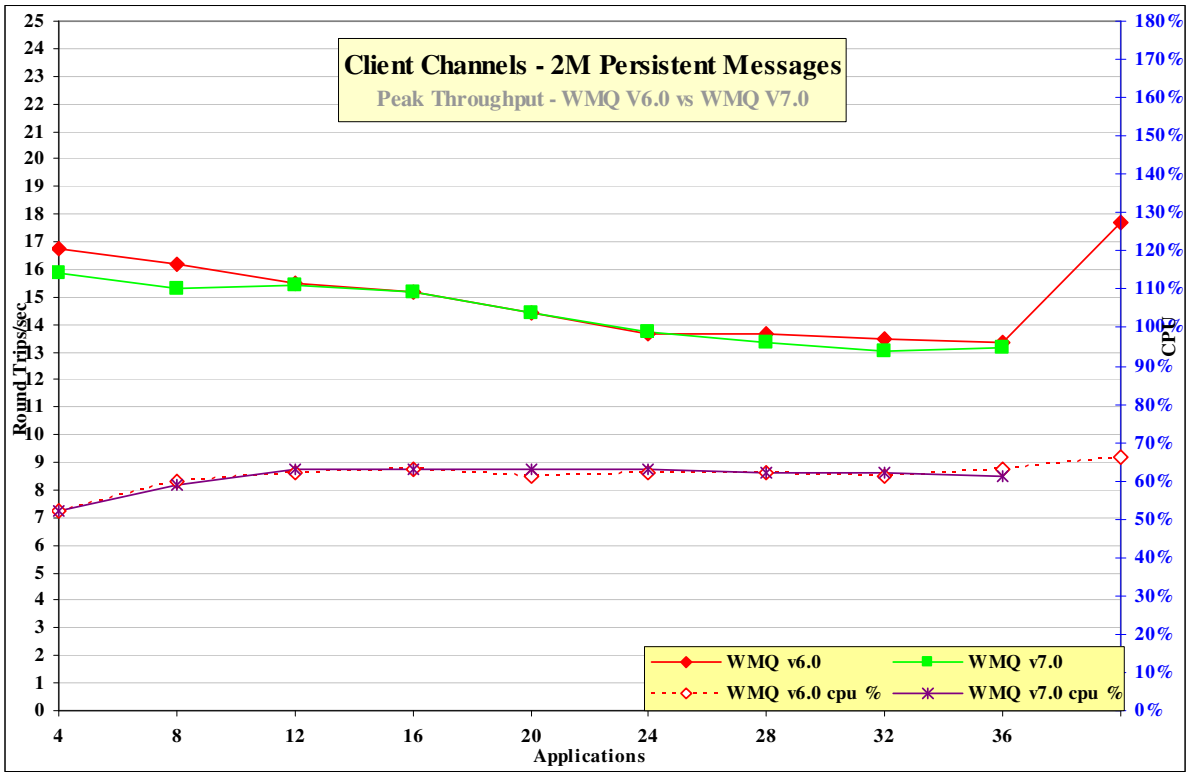


Figure 36 – 2M persistent messages, client channels

Fig 36 and **Table 24** show that the throughput of persistent messages is similar when comparing Version 6 to Version 7.

Test name: **clpm_2M**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**4**	**16.7**	**0.236**	**52%**
WebSphere MQ V7.0	**4**	**15.9**	**0..249**	**52%**

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 throughput7.

3.4.3 Distributed Queuing

Fig 37 and **fig 38** 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

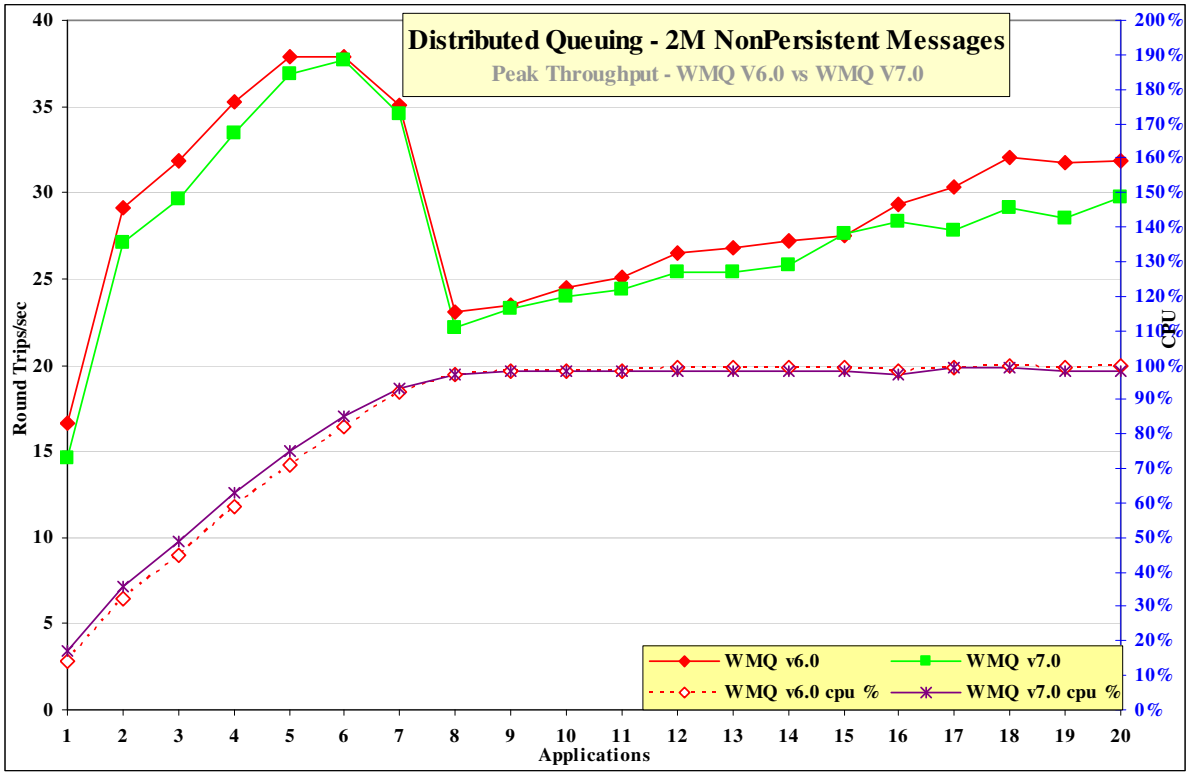


Figure 37 – 2M nonpersistent messages, distributed queuing

Fig 37 and **Table 25** show that the throughput of nonpersistent messages has degraded by 5% when comparing Version 6 to Version 7 although the CPU cost per message has increased.

Test name: **dqnp_2M**	Apps	Round Trips/sec	Response time (s)	CPU
WebSphere MQ V6.0	**5**	**37.9**	**0.152**	**24%**
WebSphere MQ V7.0	(5) **6**	(36.9) **37.7**	(0.159) **0.186**	(75%) **85%**

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

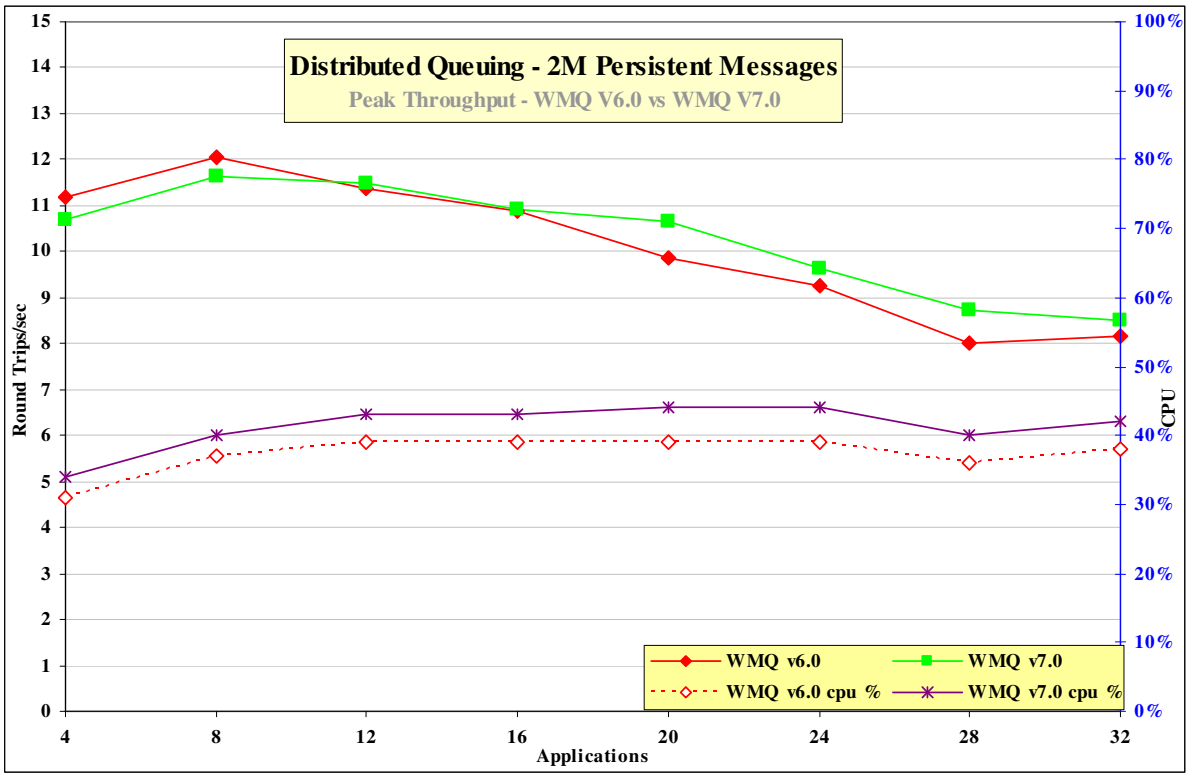


Figure 38 - 2M persistent messages, distributed queuing

and **Table 26** show that the throughput of nonpersistent messages has increased by 2% when comparing Version 6 to Version 7.

Test name: **dqpm_2M**	**Apps**	**Round Trips/sec**	**Response time (s)**	**CPU**
WebSphere MQ V6.0	**8**	**12.0**	**0.737**	**37%**
WebSphere MQ V7.0	**8**	**11.7**	**0.755**	**40%**

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 throughput7.

3.5 SSL Clients

This section uses Non-Persistent and Persistent messages of lengths 2K bytes, 20K bytes, 64K bytes and 200K bytes with no SSL cipher together with three specific ciphers namely null_sha, tls_rsa_with_aes_256_cbc_sha, and triple_des_sha_us. The ssl cipher names are abbreviated to nullsha, rsa, and 3des respectively in the graphs.

3.5.1 Non Persistent 2K byte message

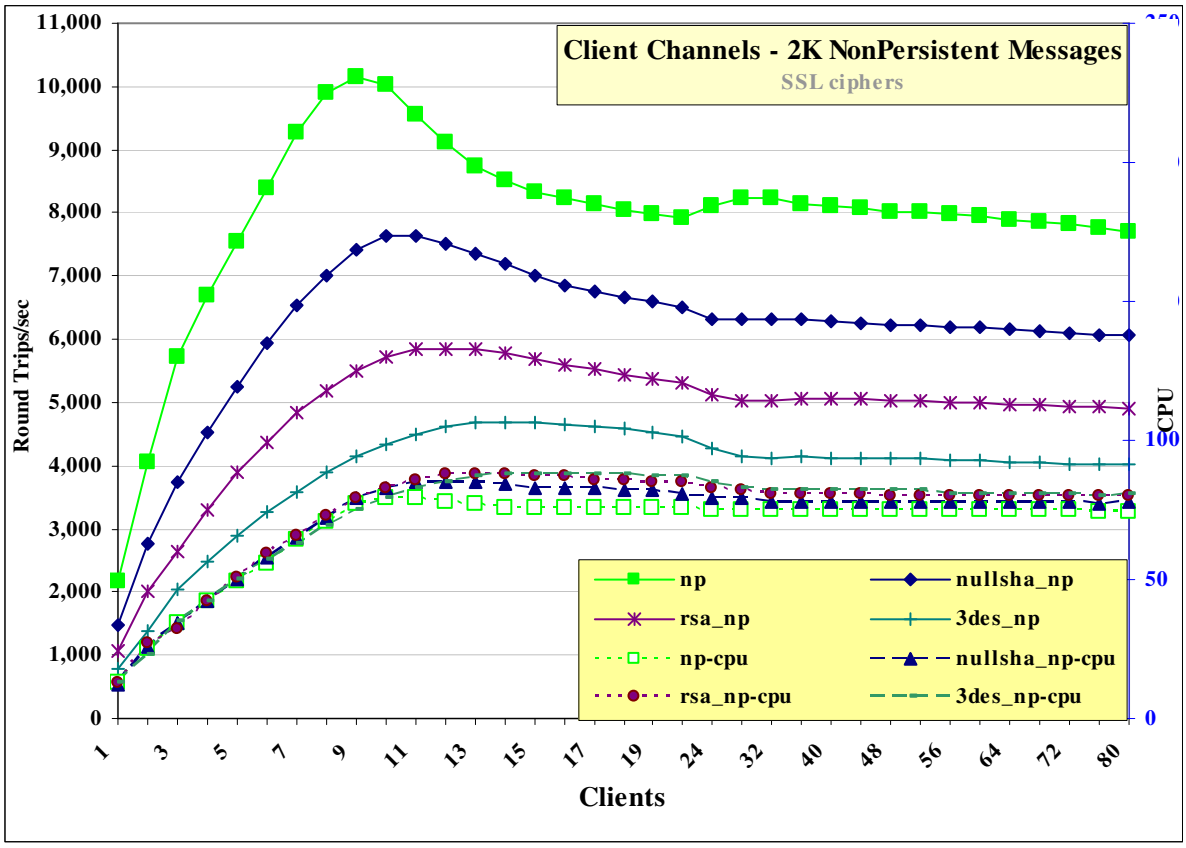


Figure 39 2K non-persistent message

	Throughput(20 clients)	cpu	Response time(secs)
CLNP(no ssl)	7932	76	0.003
null-sha	6493	81	0.004
tls_rsa_with_aes_256_cbc_sha	5313	85	0.005
triple_des_sha_us	4475	87	0.005

The cpu utilisation is similar so the relative throughput shows the cost of ssl processing

3.5.2 Non Persistent 20K byte message

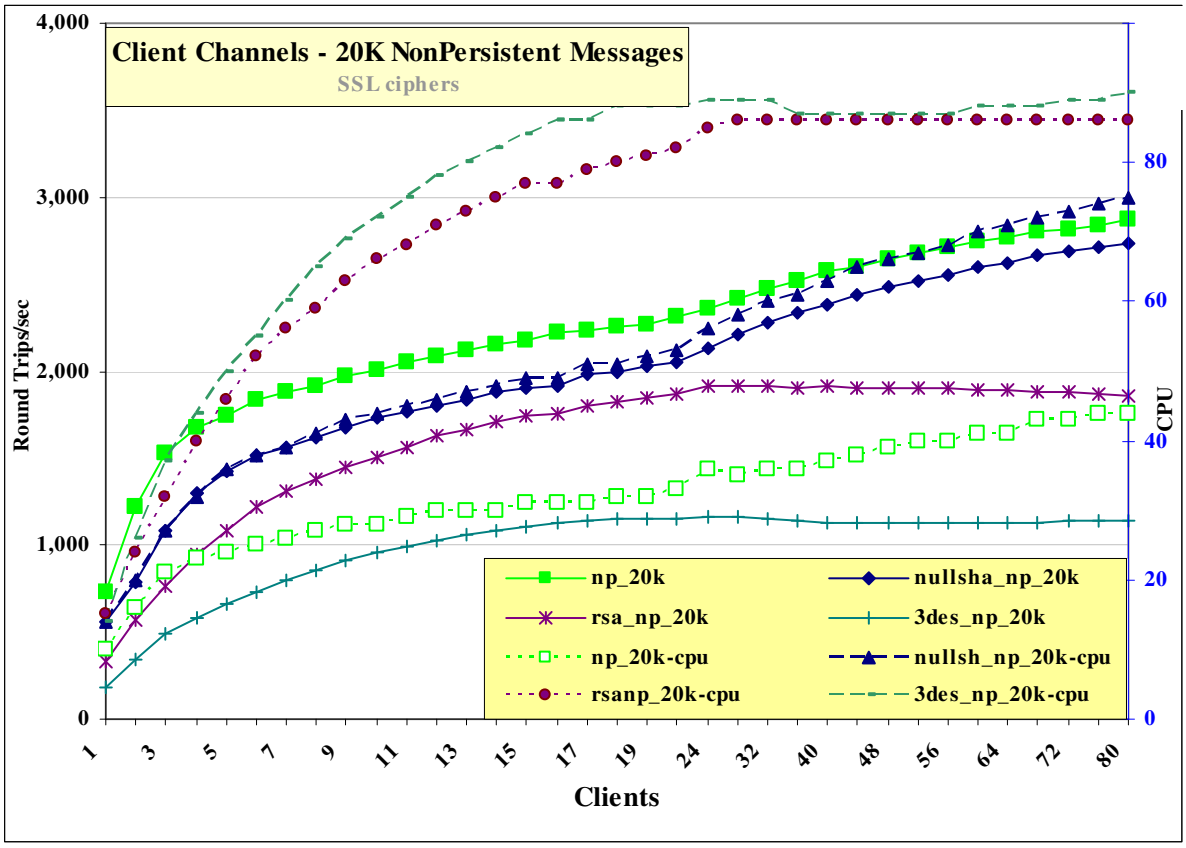


Figure 40 20K non-persistent message

	Throughput(20 clients)	cpu	Response time(secs)
CLNP(no ssl)	2309	33	0.011
null-sha	2053	53	0.011
tls_rsa_with_aes_256_cbc_sha	1863	82	0.012
triple_des_sha_us	1153	88	0.020

The RSA and 3DES lines are constrained by cpu whereas the no-ssl and null_sha have increased throughput beyond 80 clients

3.5.3 Non Persistent 64K byte message

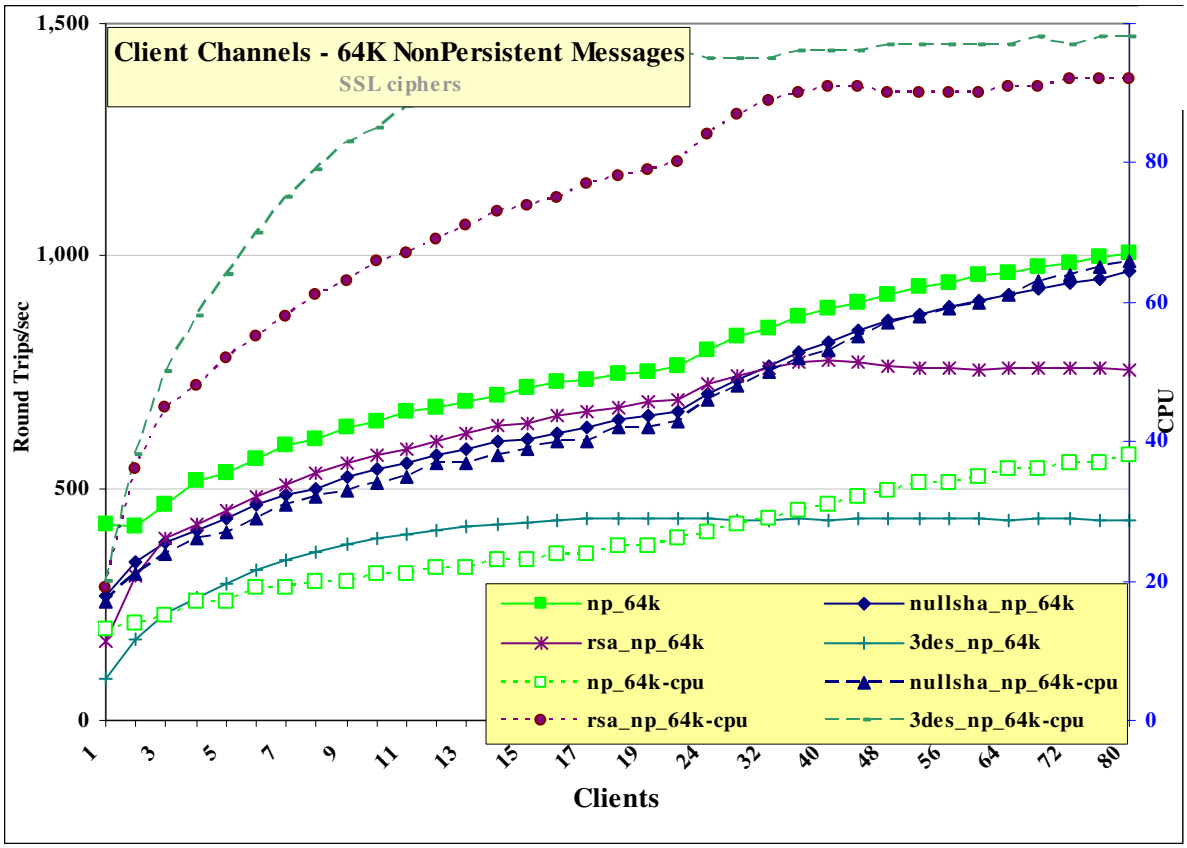


Figure 41 64K Non-persistent message

	Throughput(20 clients)	cpu	Response time(secs)
CLNP(no ssl)	761	26	0.034
null-sha	666	43	0.034
tls_rsa_with_aes_256_cbc_sha	689	80	0.034
triple_des_sha_us	434	96	0.054

The RSA and 3DES lines are constrained by CPU whereas the no-ssl and null_sha are still increasing beyond 80 clients

3.5.4 Non Persistent 200K byte message

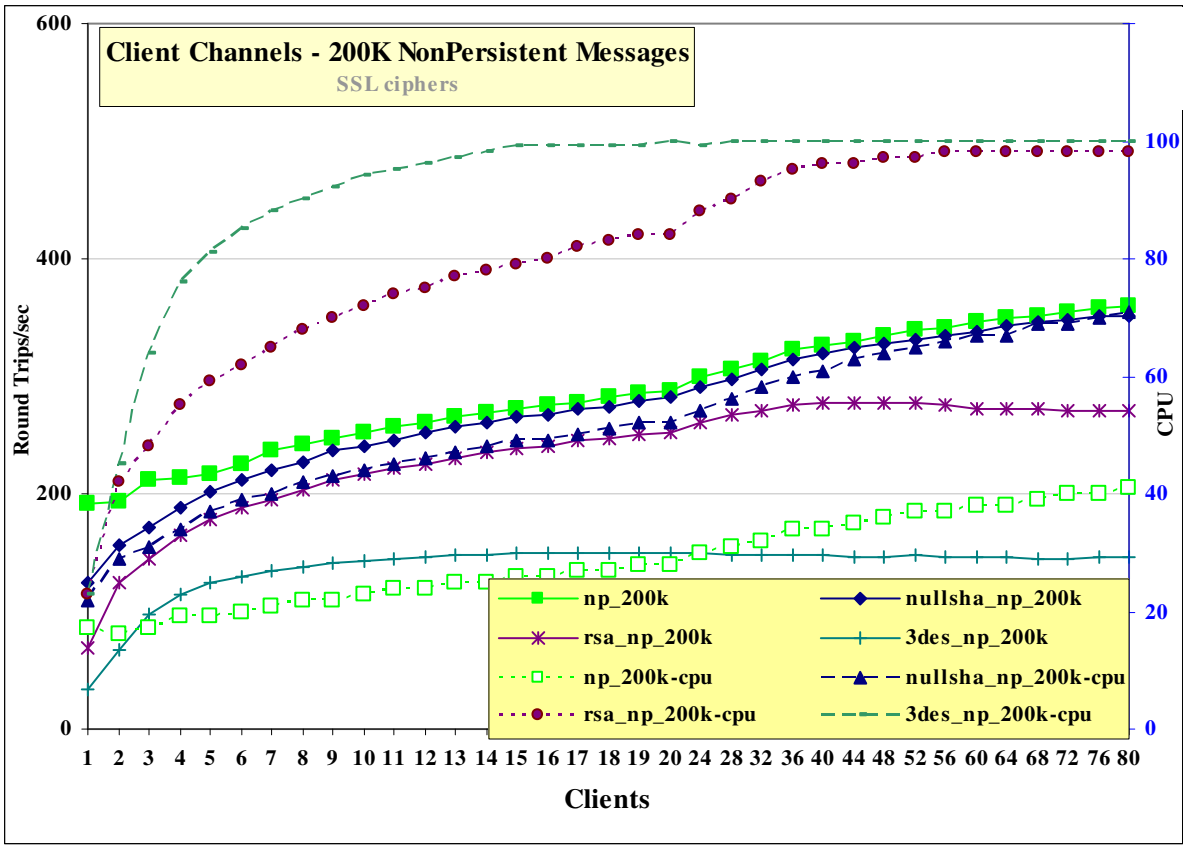


Figure 42 200k Non-persistent message

	Throughput(20 clients)	cpu	Response time(secs)
CLNP(no ssl)	287	28	0.081
null-sha	281	52	0.082
tls_rsa_with_aes_256_cbc_sha	251	84	0.091
triple_des_sha_us	149	100	0.157

The RSA and 3DES lines ar constrained by cpu but the no_ssl and nul_sha are still increasing beyond 80 clients

3.5.5 Persistent 2K byte message

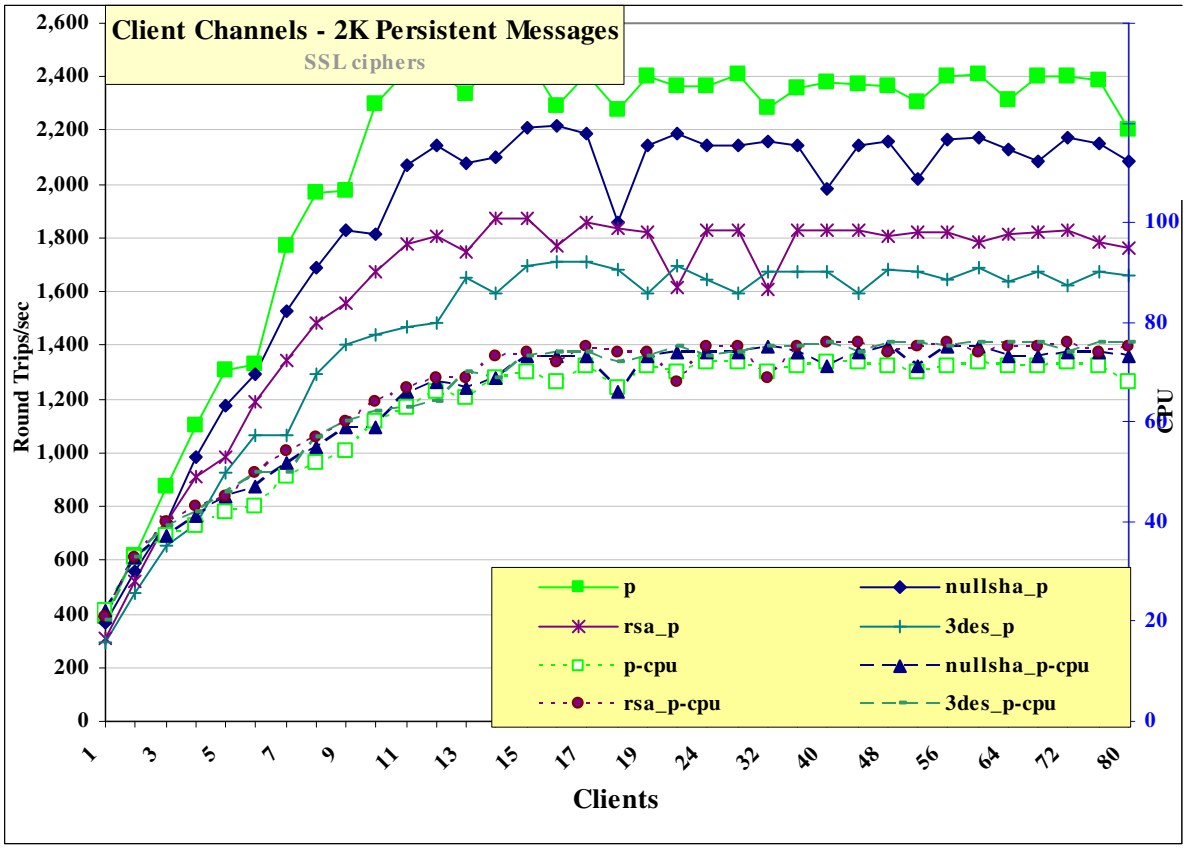


Figure 43 – 2K persistent messages

	Throughput(20 clients)	cpu	Response time(secs)
CLPM(no ssl)	2364	70	0.010
null-sha	2191	74	0.011
tls_rsa_with_aes_256_cbc_sha	1616	68	0.015
triple_des_sha_us	1697	75	0.015

3.5.6 Persistent 20K byte message

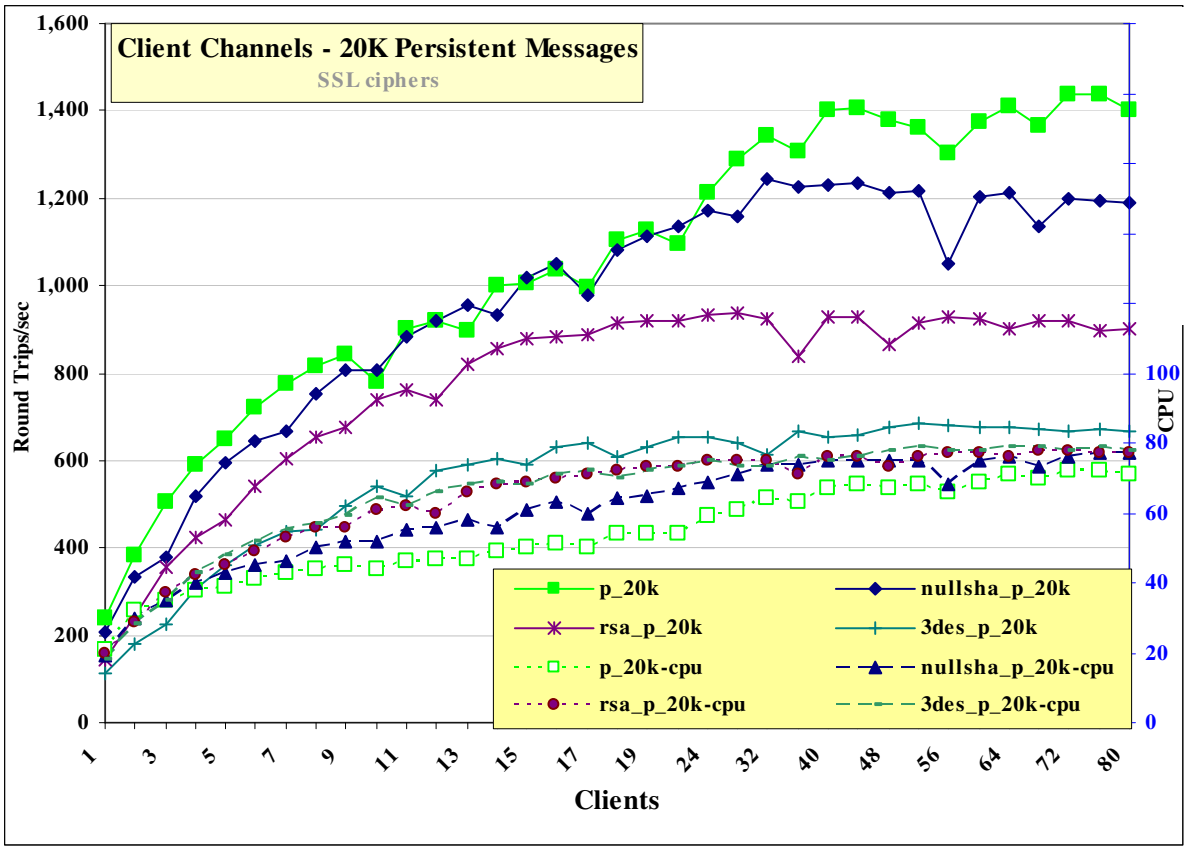


Figure 44 – 20K persistent messages

	Throughput(20 clients)	cpu	Response time(secs)
CLPM(no ssl)	1093	54	0.021
null-sha	1135	67	0.023
tls_rsa_with_aes_256_cbc_sha	918	73	0.024
triple_des_sha_us	653	73	0.036

3.5.7 Persistent 64K byte message

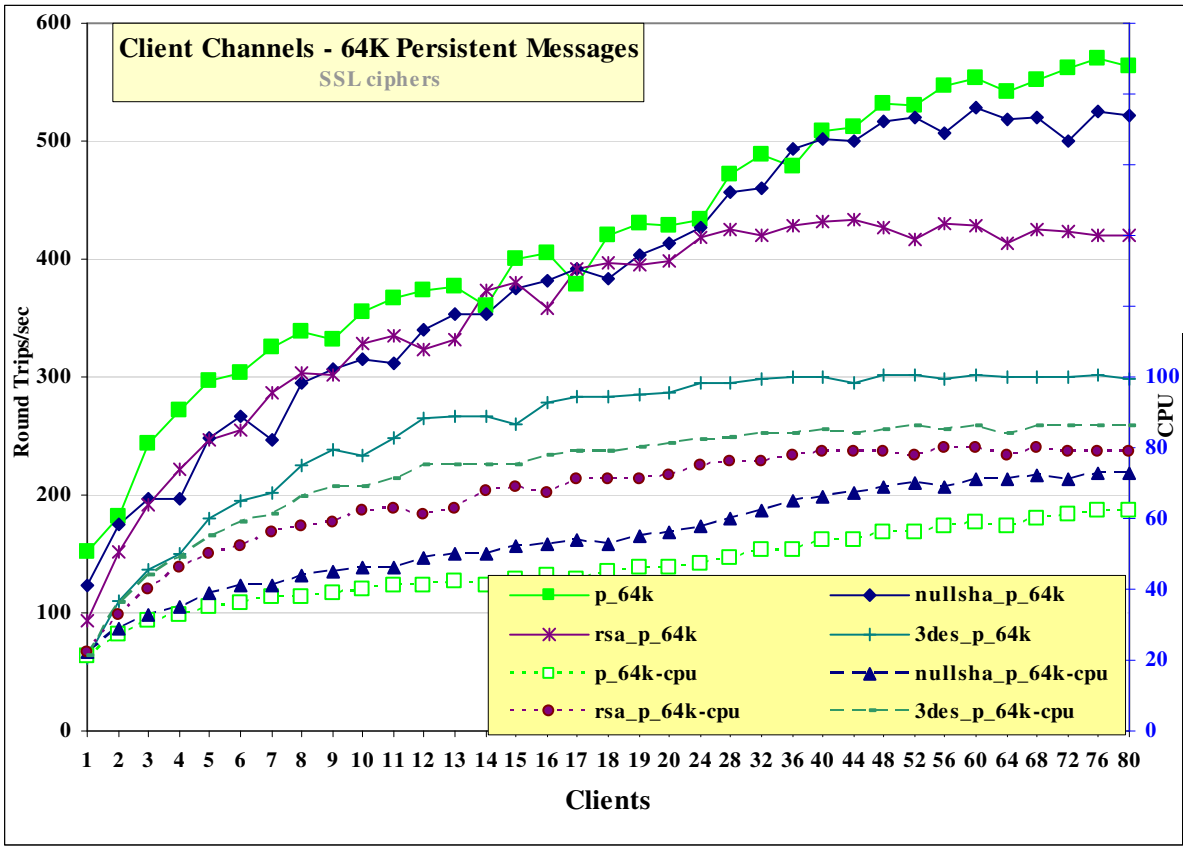


Figure 45 – 64K persistent messages

	Throughput(20 clients)	cpu	Response time(secs)
CLPM(no ssl)	428	46	0.056
null-sha	413	56	0.057
tls_rsa_with_aes_256_cbc_sha	398	72	0.059
triple_des_sha_us	287	81	0.083

3.5.8 Persistent 200K byte message

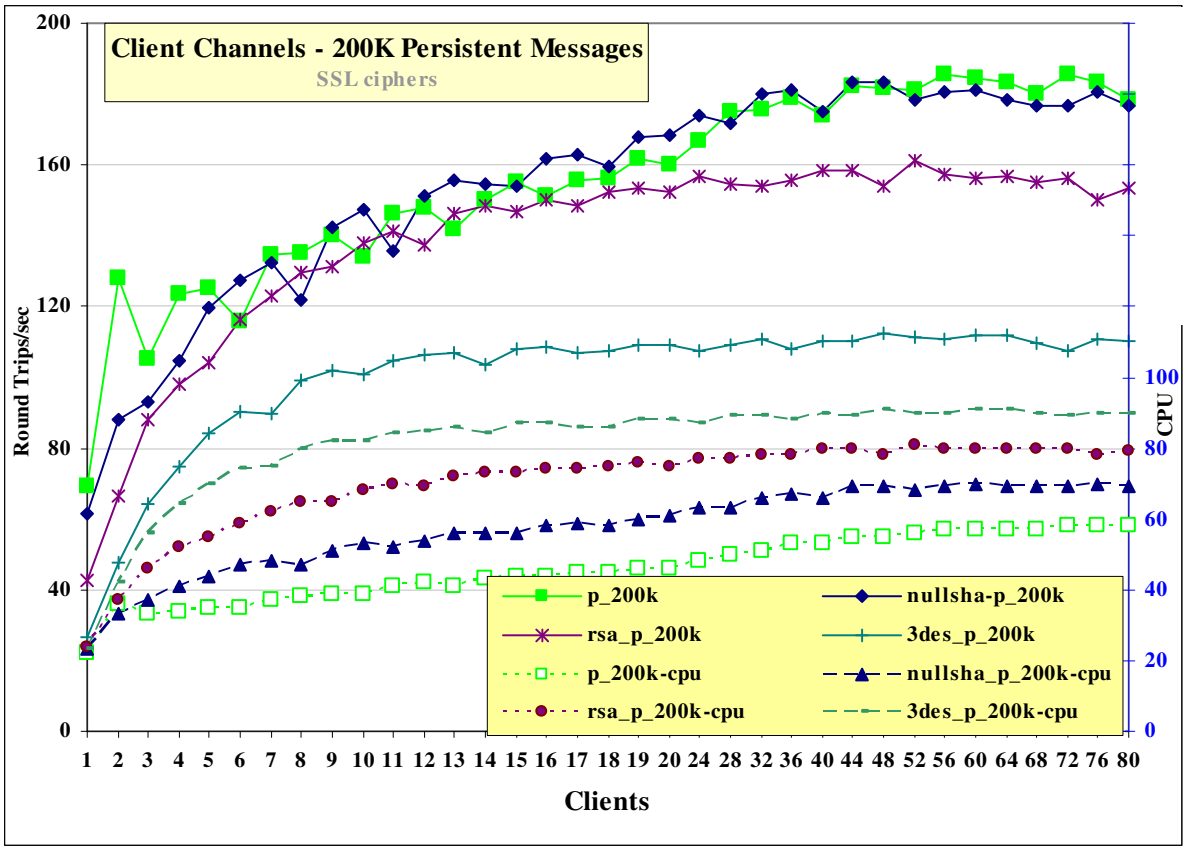


Figure 46 – 200K persistent messages

	Throughput(20 clients)	cpu	Response time(secs)
CLPM(no ssl)	160	46	0.149
null-sha	168	61	0.142
tls_rsa_with_aes_256_cbc_sha	152	75	0.153
triple_des_sha_us	109	88	0.220

3.5.9 SSL Summary

The Client SSL measurements show that ssl ciphers degrade throughput and increase cpu cost per message.

	Throughput	Cpu cost - Persist message	Cpu cost -Non-Per message
Normal(no ssl)	1	1	1
null-sha	0.9	1.2	1.7
tls_rsa_with_aes_256_cbc_sha	0.78	1.5	2.7
triple_des_sha_us (non persist)	0.44		4.3
triple_des_sha_us (Persist)	0.63	3.5	

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 47 and **Figure 48** 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

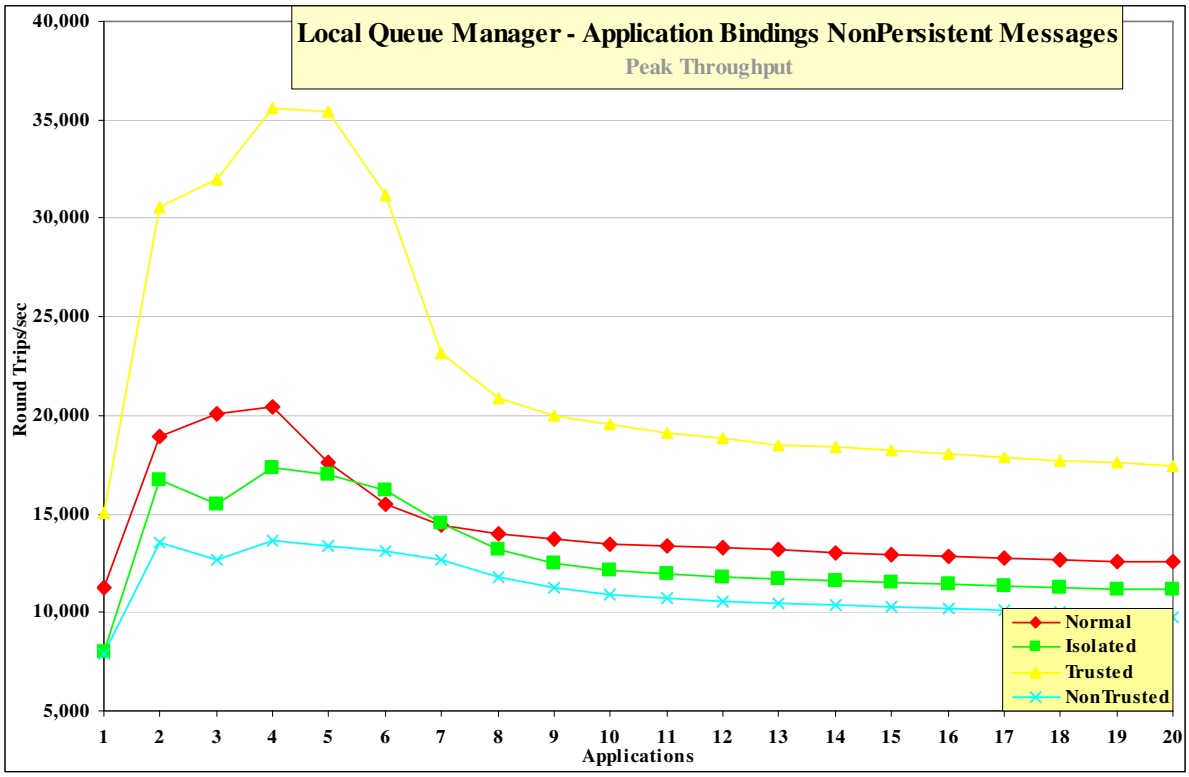


Figure 47 – Application binding, nonpersistent messages, local queue manager

Figure 47 and **Table 27** show that the throughput of nonpersistent messages when comparing Normal, Isolated, Trusted and Shared bindings.

Test	Apps	Round Trips/sec	Response time (s)	CPU
Normal	**4**	**20400**	**0.001**	**85%**
Isolated	**4**	**17374**	**0.001**	**92%**
Trusted	**4**	**35592**	**0.001**	**92%**
Shared	**4**	**13606**	**0.001**	**94%**

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

4.1.2 Persistent Messages

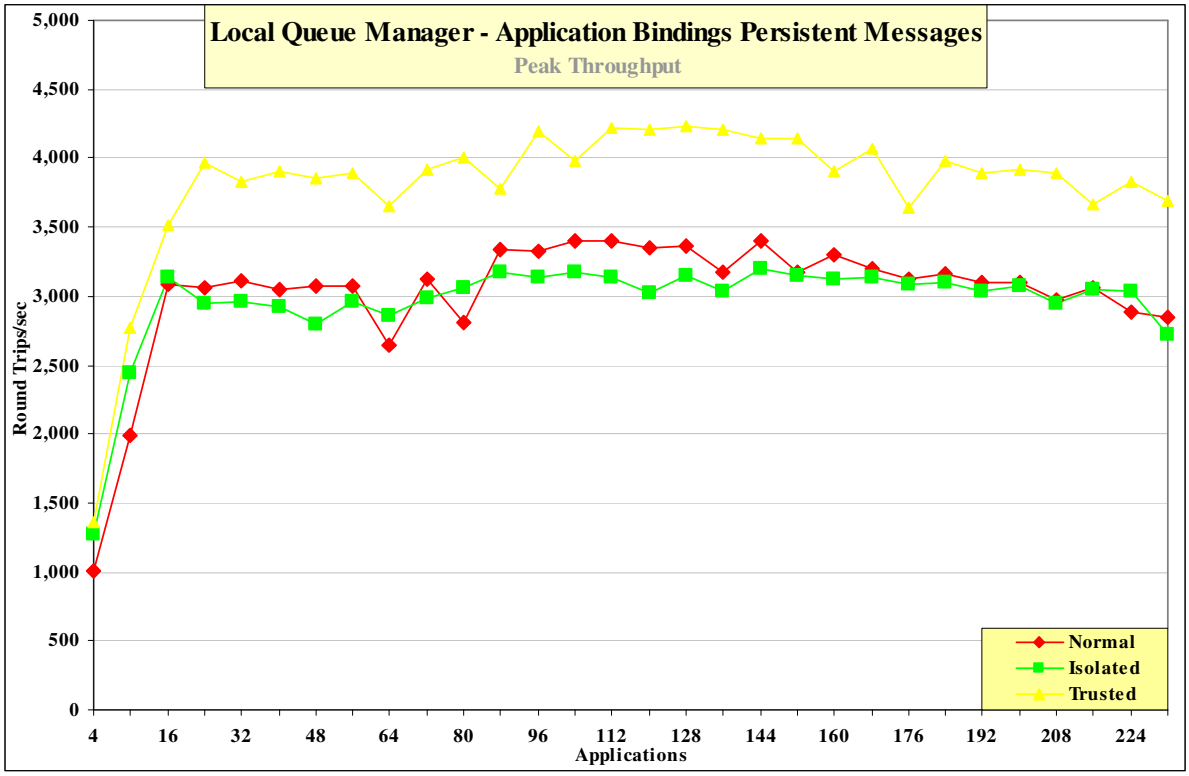


Figure 48 – Application binding, persistent messages, local queue manager

Figure 48 and **Table 28** show that the throughput of persistent messages when comparing Normal, Isolated and Trusted bindings.

Test	Apps	Round Trips/sec	Response time (s)	CPU
Normal	**144**	**3403**	**0.053**	**70%**
Isolated	**144**	**3197**	**0.053**	**73%**
Trusted	(144) **128**	(4148) **4232**	(0.043) **0.035**	(66%) **66%**

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.

.

4.2 Client Channels

Figure 49 and **Figure 50** show the nonpersistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

4.2.1 Nonpersistent Messages

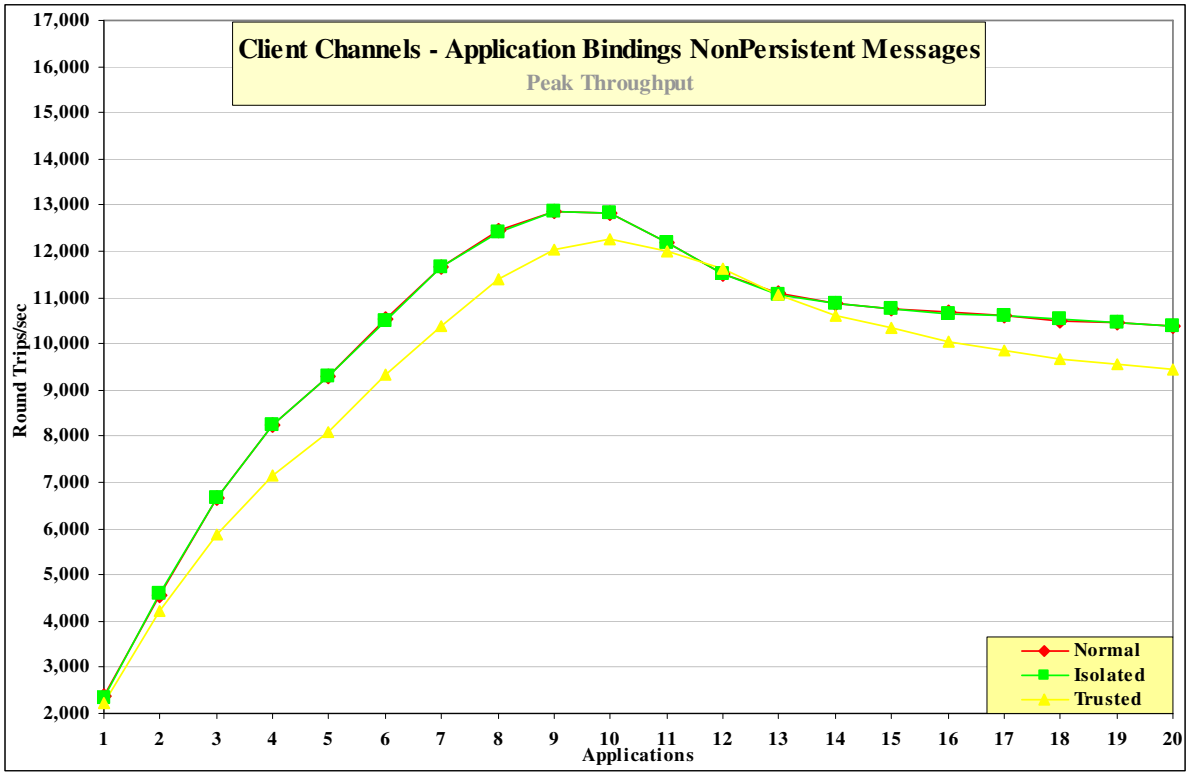


Figure 49 – Application binding, nonpersistent messages, client channels

Figure 49 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	**9**	**12862**	**0.001**	**83%**
Isolated	**9**	**12850**	**0.001**	**83%**
Trusted	**10**	**12277**	**0.001**	**34%**

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 throughput7.

4.2.2 Persistent Messages

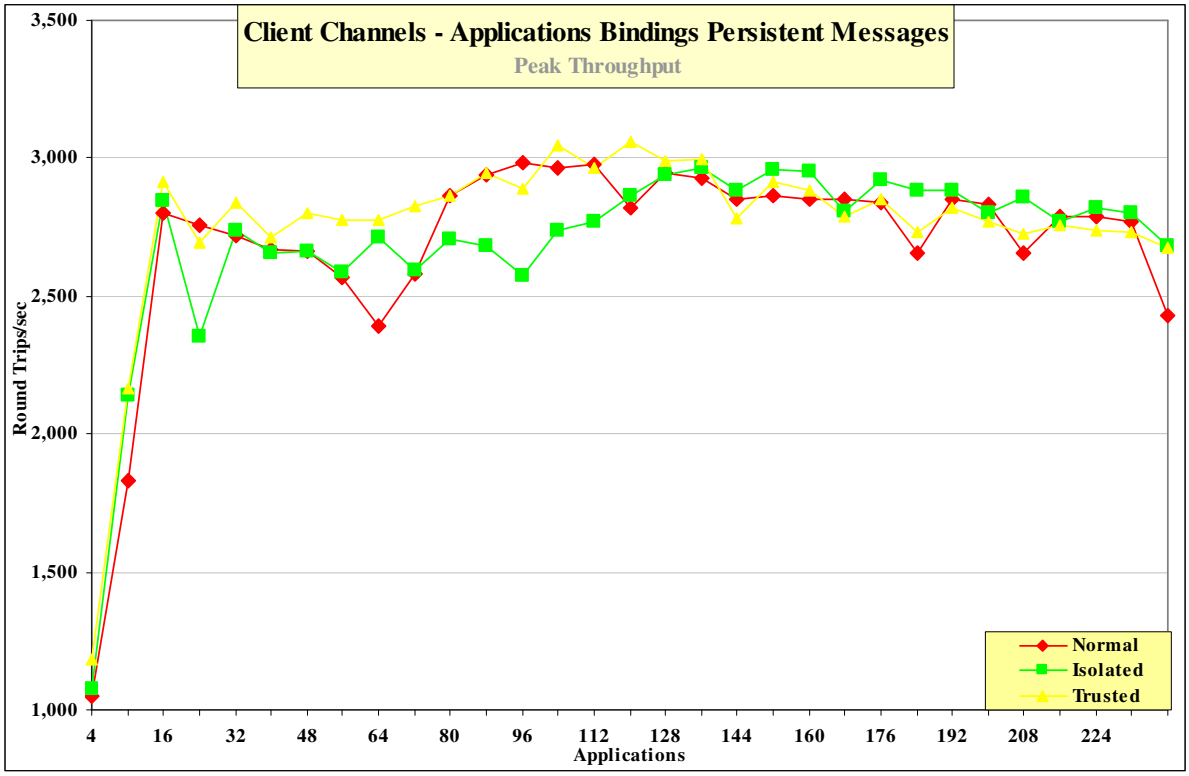


Figure 50 – Application binding, persistent messages, client channels

Figure 50 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	(120) **96**	(2821) **2986**	(0.055) **0.039**	(71%) **74%**
Isolated	**136** (120)	**2964** (5816)	**0.055** (0.0281)	**74%** (64%)
Trusted	**120**	**3057**	**0.046**	**72%**

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 50 and **Figure 51** 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

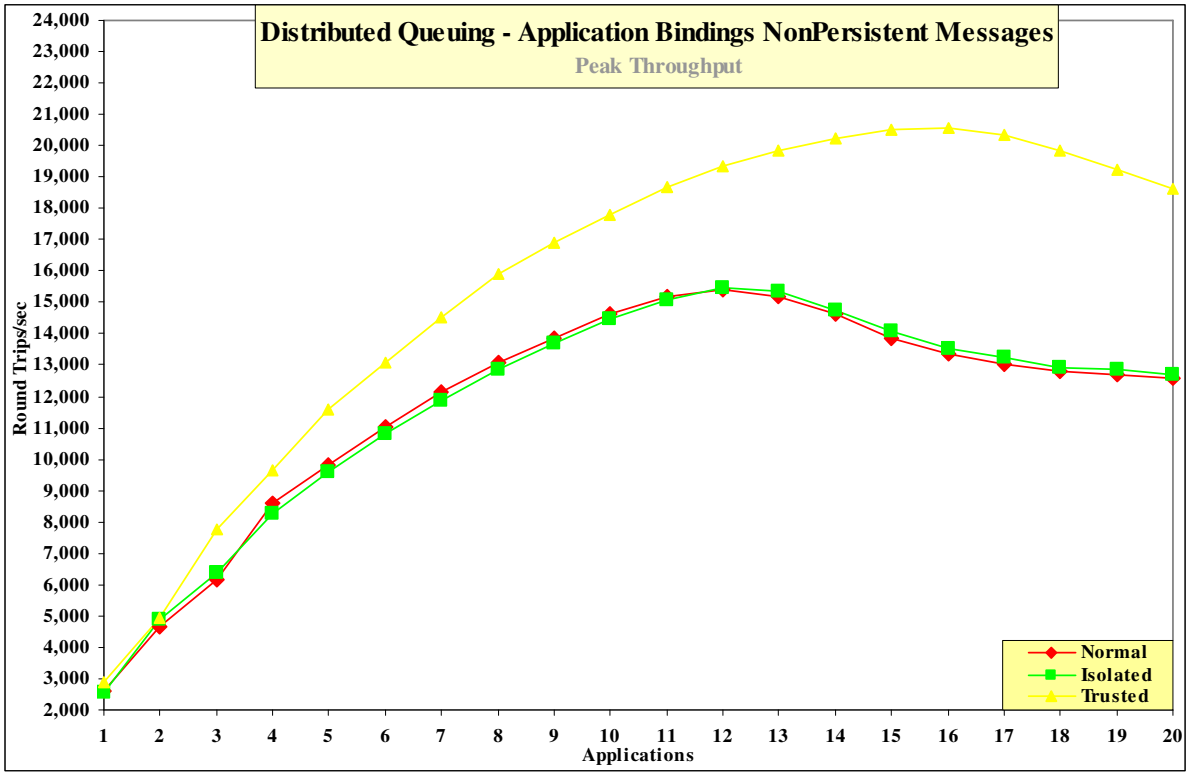


Figure 51 – Application binding, nonpersistent messages, distributed queuing

Figure 51 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	**12**	**15425**	**0.001**	**84%**
Isolated	**12**	**15426**	**0.001**	**70%**
Trusted	**16**	**20569**	**0.001**	**65%**

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

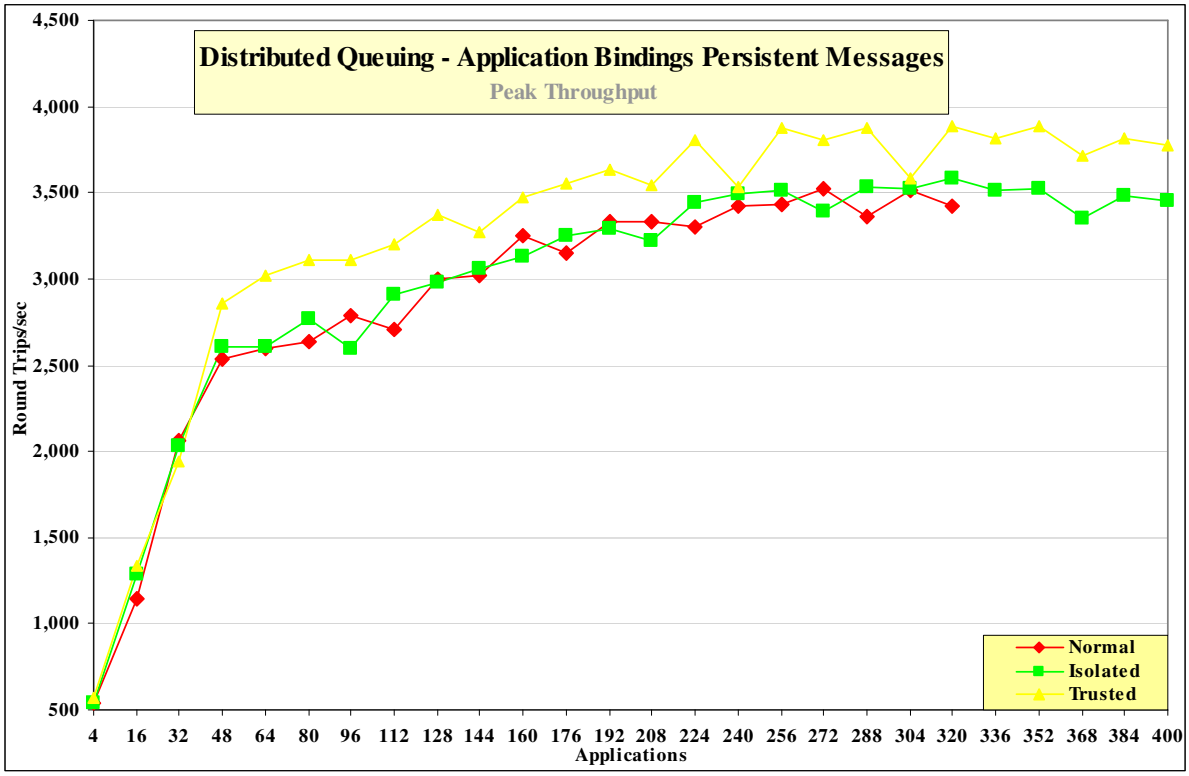


Figure 52 – Application binding, persistent messages, distributed queuing

Figure 52 and **Table 32** 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	**272**	**3524**	**0.103**	**61%**
Isolated	**320**	**3580**	**0.098**	**66%**
Trusted	**320**	**3889**	**0.095**	**66%**

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 (refer to **Figure 53** for an illustration).

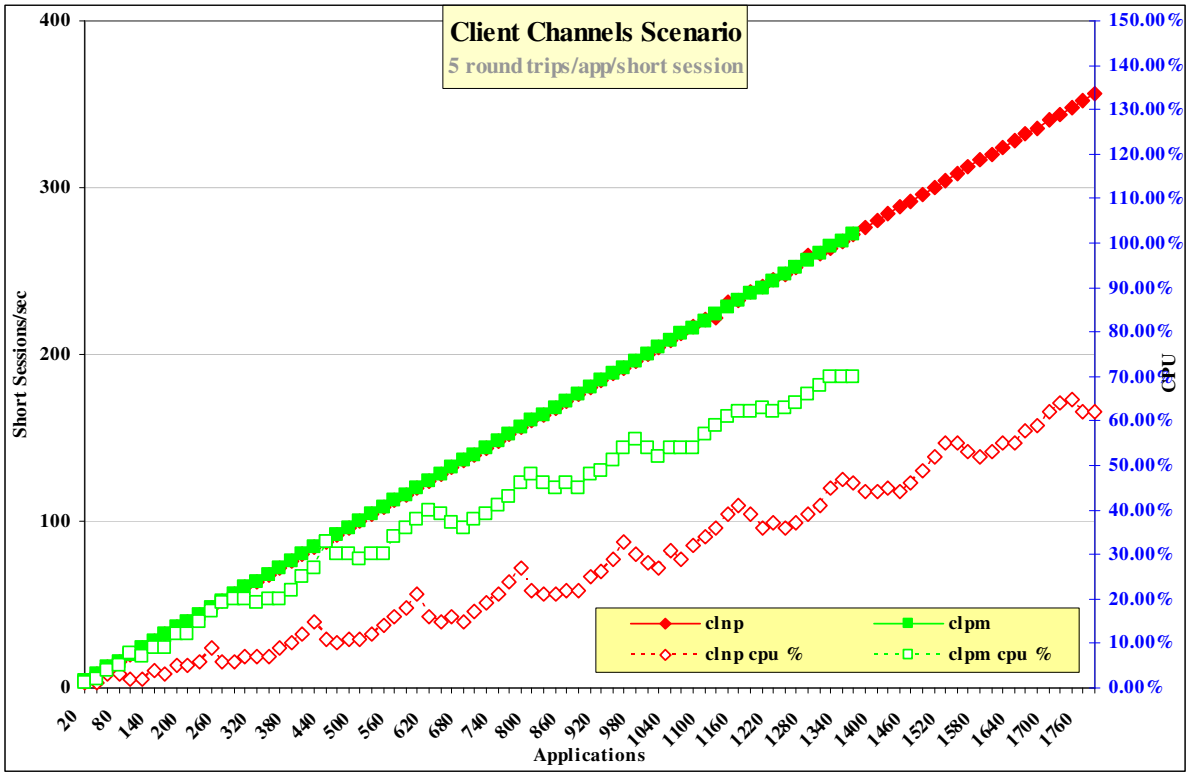


Figure 53 – Short sessions, client channels

Test name	Apps	Round Trips/sec	Short Sessions per second	Response Time (s)	CPU%
clnp_r3600	**6,600**	**6588**		**0.007**	**61%**
clnp_ss	**1780**	**1780**	**356**	**0.06**	**62%**
clpm_r3600	**2,250**	**2249**		**0.007**	**63%**
clpm_ss	**1360**	**1360**	**272**	**0.01**	**70%**

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 'runmqlsr' 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

These capacity measurements were made with MQ V7.0.1

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*. 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

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
clnp	10	n/a*	10,638	0.001	79%
clnp_r3600	6,600	3,600	6,588	0.007	61%
clnp_cmax	16000	60	263	0.001	4%
clnp_cmax_no_correllid	12800 17500&	60	213 290	0.001 0.12	9% 71%
clnp_c6000	6000	3110	5525	0.05	77%
cl_persist_c6000	6000	180	267	0.005	15%

Table 34 – Capacity measurements, client channels

- ** There was no delay between the response to the previous message and the insertion of the next message with 5 clients.*
- *& Some memory overcommitment was necessary to support the 17500 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 of memory objects results in a larger CPU cost to handle each message.
The (clnp_cmax) test uses a Get by Correlation_Id from a common reply queue for all the clients so there is a single Server input queue and a single reply to queue. Each additional Client needs a thread in the AMQRMPPA process and using a separate queue per client needs additional memory per client. These measurements use MQIBINDTYPE =FASTPATH but the default MQIBINDTYPE =STANDARD need an additional 76K mainly due to the additional thread in the AMQZLAA0 process

Test name:	Apps	Swap	Free mem	Free per Applic
clnp_cmax	1000 16000	0 0	328077 28860	217K
clnp_cmax(sharecnv=0)	500 23500 30000	0 0 348016	3538252 1542	153K
clnp_cmax_no_correlid	1000 12800 18000	924 26776 2571188	3338204 32492 18972	280K
clnp_c6000	1000 6000	0 0	3592956 2247420	269K
cl_persist_c6000	1000 6000	12316 12316	3246156 2033220	242K

These storage calculations are for clients that have a separate IP socket for each MQ connection (sharecnv=1). 10 MQ connects from the same application process (default) will reduce the cost per client by 90K bytes. Using V6 compatability mode (sharecnv=0) will reduce the cost per client by 70K bytes.

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=16000, LogPrimaryFiles=3,

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
dqnp	40	n/a*	16045	0.003	56%
dqnp_r3600	4600	3600	4598	0.001	30%
dqnp_q1000	1000	20000	5534	0.002	37%
dq_persist_q1000	1000	5080	1349	0.036	52%
dqnp_qmax	6300@	60	105	0.01	4%
dq_persist_qmax	6100@	60	101	0.05	15%

Table 35 – Capacity measurements, server channels

- * *There was no delay between the response to the previous message and the insertion of the next message with 40 driving applications..*
- @ The alternative Driver machine with more than 4GB of memory was used

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 mangers are connected into a central hub and the following table shows the storage on the central hub.

Test name:	Apps	Swap	Free mem	Free per Applic
dqnp_q1000	100 1000	176	2353228 2011772	388K
dqnp_q1000	100 1000	12396 12396	3567324 3023508	604K
dq_persist_q1000	100 1000	10512 10512	1730652 1260572	522K
dqnp_qmax	100 1800	12396 12396	2510316 1774908	433K
dq_persist_qmax	100 3100	10512 10512	1733132 97684	527K

Table 36 – DQ capacity, memory utilisation

Note: The table above show the swap memory measured at the given number of driving applications. The swap and free memory cost is the additional cost per driving application (in this test scenario this relates to the cost of an MQI-Sender/Receiver pair of channels plus Transmission queue connected on the server machine).

.

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 achieving 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 archieving 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%. The Linux measurements in this report used LogWriteIntegrity=TripleWrite

7.1.3 Channels: Process or Thread, Standard or Fastpath?

Threaded channels are used for all the measurements in this report ('runmqlsr', and for server channels an MCATYPE of 'THREAD') the threaded listener 'runmqlsr' can now be used in all scenarios with client and server channels. Additional resource savings are available using the 'runmqlsr' 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 to referred to as Compatability 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 'runmqlsr' has a much smaller overhead compared to inetd 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 Tuning the Operating System (Linux RHES)

/etc/sysctl.conf was updated with
kernel.sem = 500 256000 250 1024
kernel.msgmni = 1024
kernel.shmmni = 4096
kernel.shmall = 2097152
fs.file-max = 200000
kernel.shmmax = 268435456

7.4 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. VMSTAT reports on 'in use' and 'free' memory as seen by the operating system page manager.

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

7.4.1 Queue Manager

Starting a MQ Queue manager generated using default values reduces the FRE by 90M bytes.

7.4.2 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.

7.4.3 Client Channels

Each MQ client channel uses between 150K - 400K bytes for processing 2K byte messages depending on traffic rate (Chapter 6 of the MQ V7 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.

7.4.4 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.

7.4.5 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.

7.4.6 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 than 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.

7.4.7 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.

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 '**WebSphere MQ** V7.0 on Linux has similar performance characteristics to the V6 product. The comparisons in this report show that throughput has dropped by an average of 6% overall (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 13%.

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.

Performance Headlines' starting on **page 1**.

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 "**WebSphere MQ** V7.0 on Linux has similar performance characteristics to the V6 product. The comparisons in this report show that throughput has dropped by an average of 6% overall (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 13%.

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.

Performance Headlines" starting on **page 1**.

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 8864 4RG
Processor:	3.3GHz Intel xeon (7140N)
Architecture:	2 dual core CPU (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 8864 4RG
Processor:	3.3GHz Intel xeon
Architecture:	2 dual core CPU (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 (used for DQ_qmax)
Model:	x3850 M2 7141 4RG
Processor:	2.9GHz Intel xeon 7350
Architecture:	2 CPU quad core
Memory (RAM):	32Gb
Disk:	4 Internal SCSI (72Gb each), (O/S, /var/mqm/, var/mqm/log, swap)
Network:	1Gbit Ethernet Adapter

The MQ SAN consists of a pair of 2026 model 432 (McDATA ES-4700) switches running at 4Gb/s with 32 ports each. They are connected together via two inter-switch links to form a single SAN fabric.

The MQ hosts attach via this SAN to a DS6800 disk array (1750 model 511) with one expansion drawer.
Each drawer (controller + expansion) contains 16 x 73Gb 15K fibre channel disk drives, so there are a total of 32 physical drives.
The 32 drives are configured as four RAID-5 arrays, each of which is 6+Parity+Spare (the number of spares is defined by the configuration of the DS6800).
The controller has an effective cache size of 2.6Gb plus 0.3Gb of NVS

8.3 Software

Linux 64 bit:	Red Hat Enterprise Linux AS release 4 (Nahant Update 7)
MQSeries:	Version 7, Version 6
Compiler:	C for AIX Compiler, Version 6

9 Glossary

Test name	The name of the test. *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:* ***local*** *=> local queue manager test scenario* ***cl*** *=> client channel test scenario* ***dq*** *=> distributed queuing test scenario* ***np*** *=> nonpersistent messages* ***pm*** *=> persistent messages* ***r3600*** *=> 1 round trip per driving application per second* ***runmqlsr*** *=> channels using the 'runmqlsr' listener (client channel test scenario, in addition to 'runmqchi' for distributed queuing test scenarios)* ***c6000*** *=> 6,000 client driving applications (i.e. 6,000 MQI-client connections)* ***q1000*** *=> 1,000 server channel pairs* ***max*** *=> maximum number of channels (or channel pairs)* ***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)*
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)	The percentage increase in the total message throughput rate. *Note: The nature of the comparison is noted under each table where percentage improvements have been given.*
CPU	As reported by VMSTAT
Resp time (s)	The average response time each round trip, as measured and averaged by all the driving applications.
CURDEPTH	The number of messages on the input queue as a snapshot. *Note: runmqsc <qmname>, DISPLAY QLOCAL(<qname>) CURDEPTH*
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	Free memory as reported by IOSTAT
SC0	SHARECNV=0 specified on the def channel(x) chltype(svrconn) command. Version 6 compatibility mode
SC1	SHARECNV=1 specified on the def channel(x) chltype(svrconn) command. Separate socket per client