WebSphere MQ zLinux v7.1
Performance Evaluations

Version 1.0

December 2011

Fred Preston, Craig Stirling, Ivans Ribakovs, Peter Toghill.

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.

First Edition, December 2011.

This edition applies to *WebSphere MQ for zLinux v7.1* (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

Preface

Target audience

The SupportPac was designed for people who:

- Will be designing and implementing solutions using WebSphere MQ for zLinux v7.1.
- Want to understand the performance limits of WebSphere MQ for zLinux v7.1.
- Want to understand what actions may be taken to tune WebSphere MQ for zLinux v7.1.

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 arranged**'**—Page VI** to familiarise themselves with where specific information can be found for later reference.

The contents of this SupportPac

This SupportPac includes:

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

Feedback on this SupportPac

We welcome constructive feedback on this report.

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

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).

IBM zSeries 990 2084-331 with 4 CPUs in a VM LPAR and with 2GB of RAM was used as the Device under test.

A xSeries 3850 box containing 4 quad-core 2.93GHz Intel Xeon CPUs and 32GB of RAM was used as the Driver.

<u>How this document is arranged</u>

Pages: 1-13

The first section 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.

<u>Large Messages</u>

Pages: 18-38

The second section contains performance measurements for *large messages*. This includes *MQI response times* of 50 byte to 2MB messages. It also includes *20K, 200K* and *2M* byte messages using the same scenarios as for the 2KB messages".

<u>Application Bindings</u>

Page: 39-44

The third section contains performance measurements for *'trusted, shared, and isolated'* server applications, using the same three scenarios as for the 2KB messages.

Tuning Recommendations

Pages: 48- 49

Tuning guidance specific to v7.1 on Linux

Measurement Environment

Pages: 53 54

A summary of the way in which the workload is used in each test scenario is given in the "headlines" section. This includes a more detailed description of the workload, hardware and software specifications.

Glossary

Page: 542

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

CONTENTS

TABLES

FIGURES

1 Overview

WebSphere MQ v7.1 on zLinux has improved performance in almost every area. For 2KB messages, almost every test shows improvements over earlier versions of WMQ.

Using area under the graph performance analysis techniques v7.1 compares to previous releases as follows:-

- For 2K non-persistent messages v7.1 is 74% better than v6.0.2.11
- For 2K persistent messages v7.1 is 9% better than v6.0.2.11

- For 2K non-persistent messages v7.1 is 100% better than v7.0
- For 2K persistent messages v7.1 is 23% better than v7.0

- For 2K non-persistent messages v7.1 is 84% better than v7.0.1.6
- For 2K persistent messages v7.1 is 17% better than v7.0.1.6

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 stop automatically after the response time exceeds 1 second.

2.1 Local Queue Manager Test Scenario

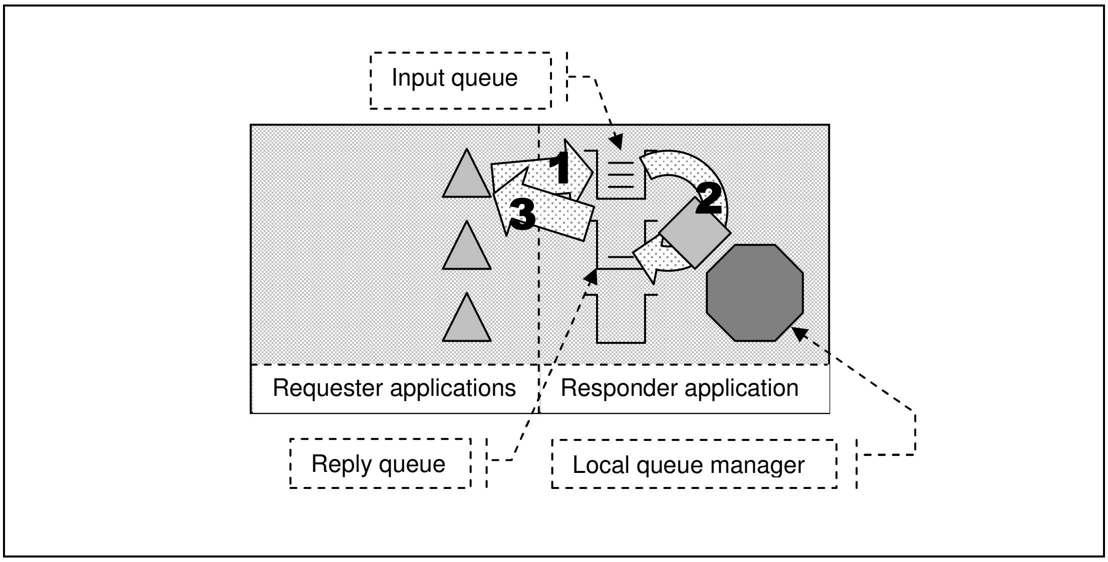


Figure 1 – Connections into a local queue manager

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.

Non-persistent and persistent messages were used in the local queue manager tests, with a message size of 2KB. 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 Non-persistent Messages – Local Queue Manager

Figure 2, Figure 3 and Figure 4 shows the non-persistent, non-persistent non-trusted 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) for different production levels of WebSphere MQ (versions 7.1, 7.0.1.6, 7.0 and 6.0.2.11).

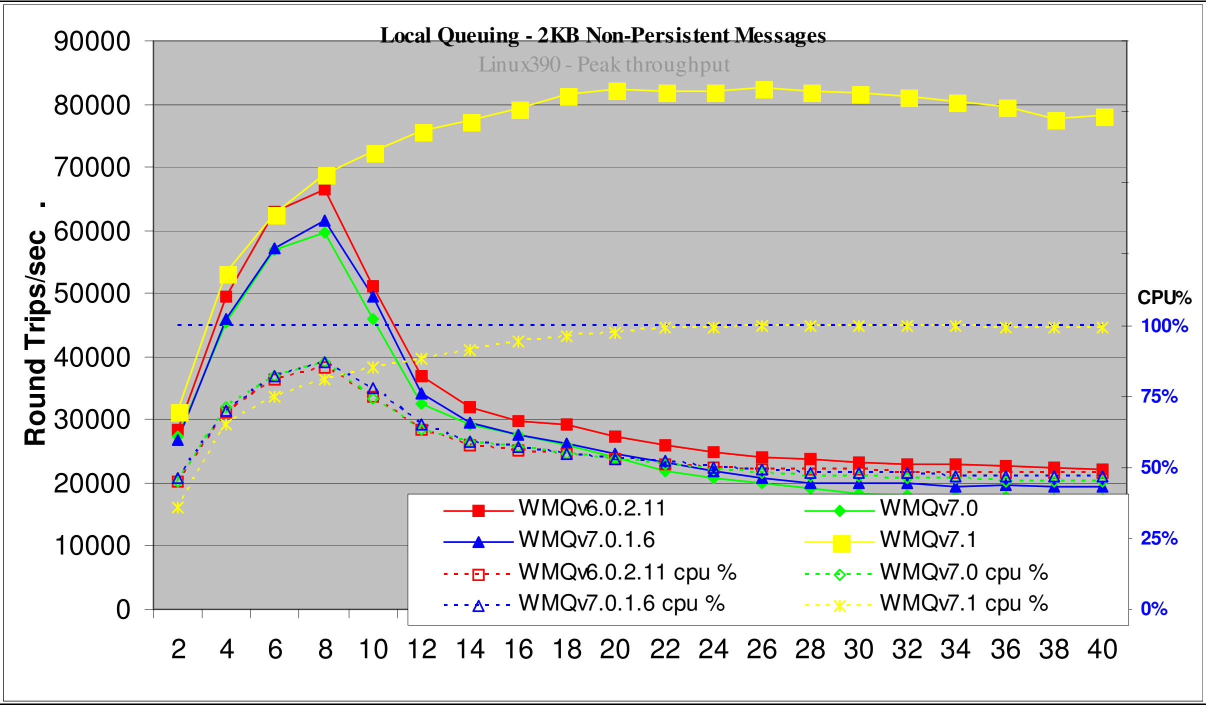


Figure 2 – Performance headline, non-persistent messages and local queue manager.

Figure 2 and Table 1 show that the peak throughput of non-persistent messages has increased by 35% when comparing version 7.1 to 7.0.1.6 and 24% when comparing version 7.1 to 6.0.2.11.

Test Name: Local Queuing - 2KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	8	66459	0.00012	85%
WMQv7.0	8	59678	0.00016	87%
WMQv7.0.1.6	8	61431	0.00016	87%
WMQv7.1	26	82671	0.00026	100%

Table 1 – Performance headline, non-persistent messages and local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.1.2 Non-persistent Messages – Non-trusted – Local Queue Manager

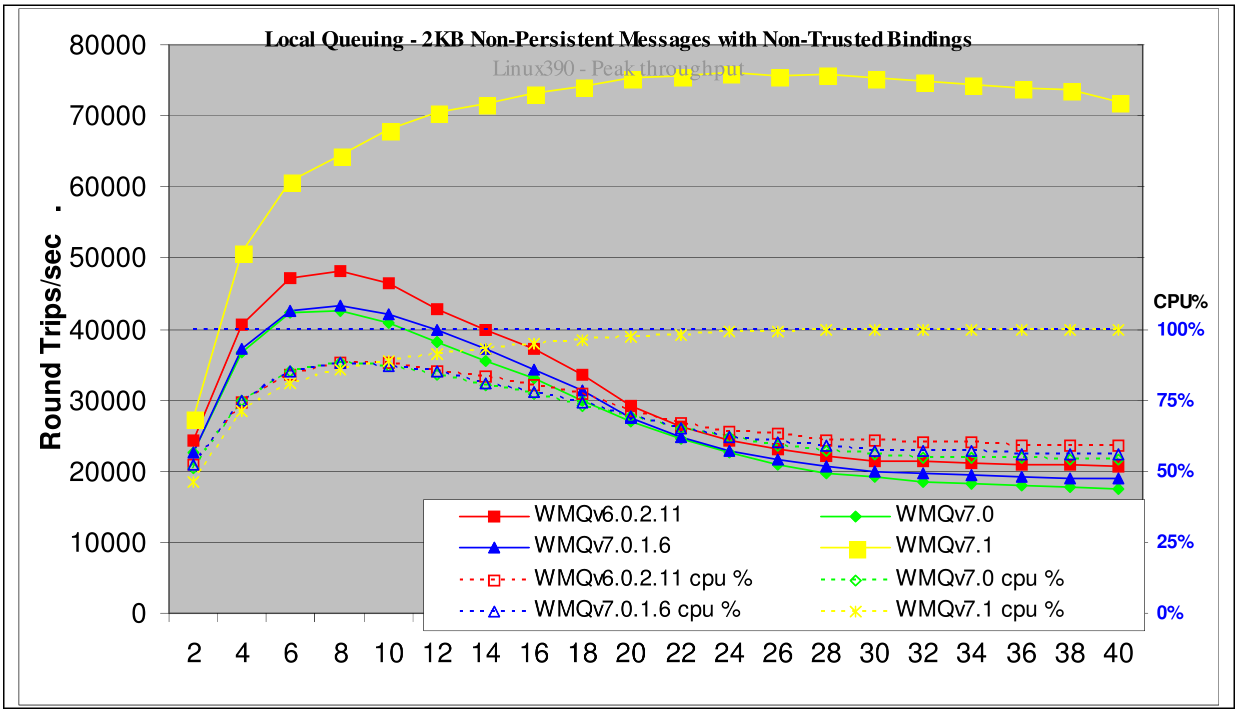


Figure 3 - Performance headline, non-persistent, non-trusted messages and local queue manager.

Figure 3 and Table 2 shows that the peak throughput of non-persistent, non-trusted messages (shared bindings - MQIBINDTYPE=NORMAL) has increased by 76% when comparing version 7.1 to 7.0.1.6 and 58% when comparing version 7.1 to 6.0.2.11.

Test Name: Local Queuing - 2KB Non-Persistent Messages with Non-Trusted Bindings	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	8	48211	0.00017	88%
WMQv7.0	8	42642	0.0002	88%
WMQv7.0.1.6	8	43306	0.00019	88%
WMQv7.1	24	76042	0.00027	99%

Table 2 – Performance headline, non-persistent, non-trusted messages and local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.1.3 Persistent Messages – Local Queue Manager

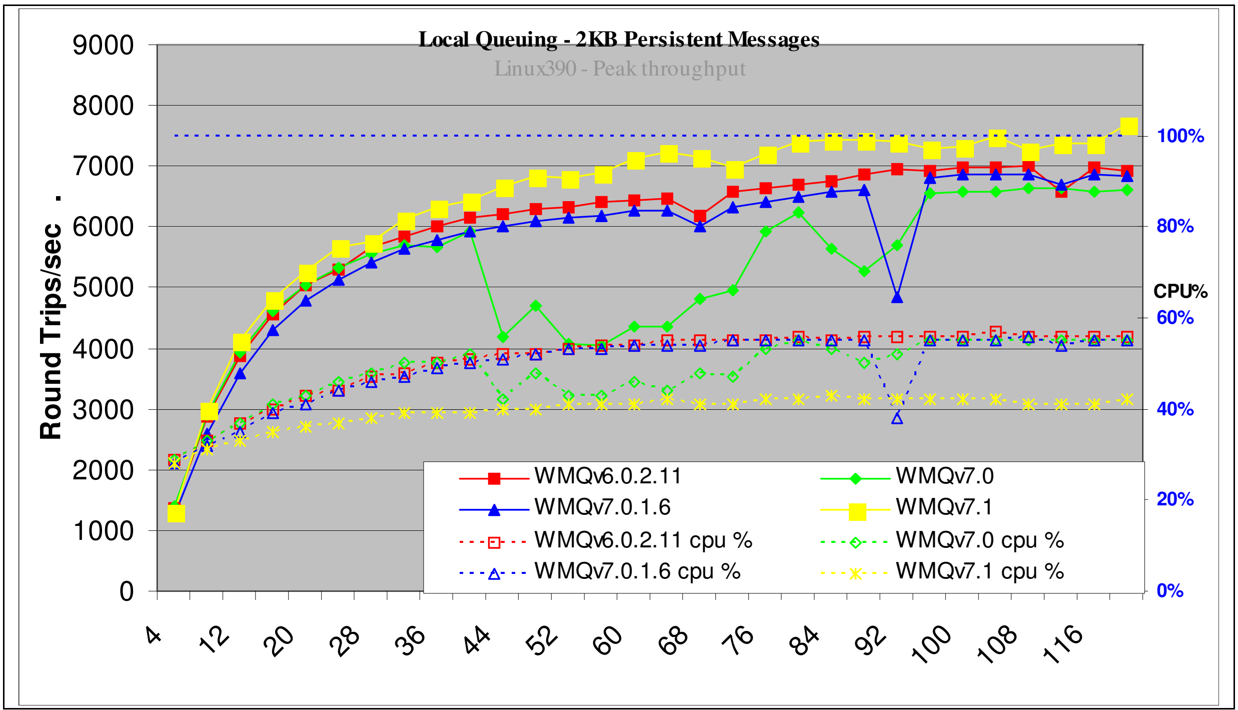


Figure 4 – Performance headline, persistent messages and local queue manager

Figure 4 and Table 3 shows that the peak throughput of persistent messages has increased by 12% when comparing version 7.1 to 7.0.1.6 and 10% when comparing version 7.1 to 6.0.2.11.

Test Name: Local Queuing - 2KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	108	6995	0.018	56%
WMQv7.0	112	6624	0.02	55%
WMQv7.0.1.6	104	6867	0.017	55%
WMQv7.1	120	7689	0.018	42%

Table 3 – Performance headline, persistent messages and local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.2 Client Channels Test Scenario

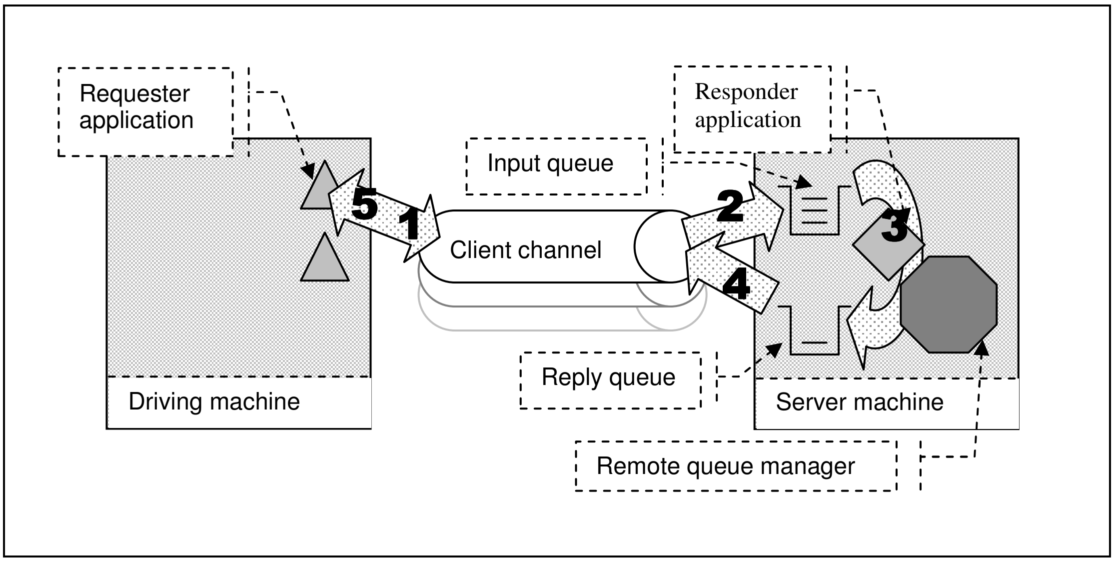


Figure 5 – MQI-client channels into a remote queue manager

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.

Non-persistent and persistent messages were used in the client channel tests, with a message size of 2KB. 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 onwards will multiplex multiple clients from the same process over one TCP socket. We have standardized all client measurements to use SHARECNV(1) since we have various tests that have between 1 and 100 clients per process and we are interested in results when all the clients come from different computers. Further information in section 7.1.4

2.2.1 Non-persistent Messages – Client Channels

Figure 6, **Figure 7** and Figure 8 shows the non-persistent, non-persistent non-trusted and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario (see Figure 5 on the previous page) for different production levels of WebSphere MQ (versions 7.1, 7.0.1.6, 7.0 and 6.0.2.11).

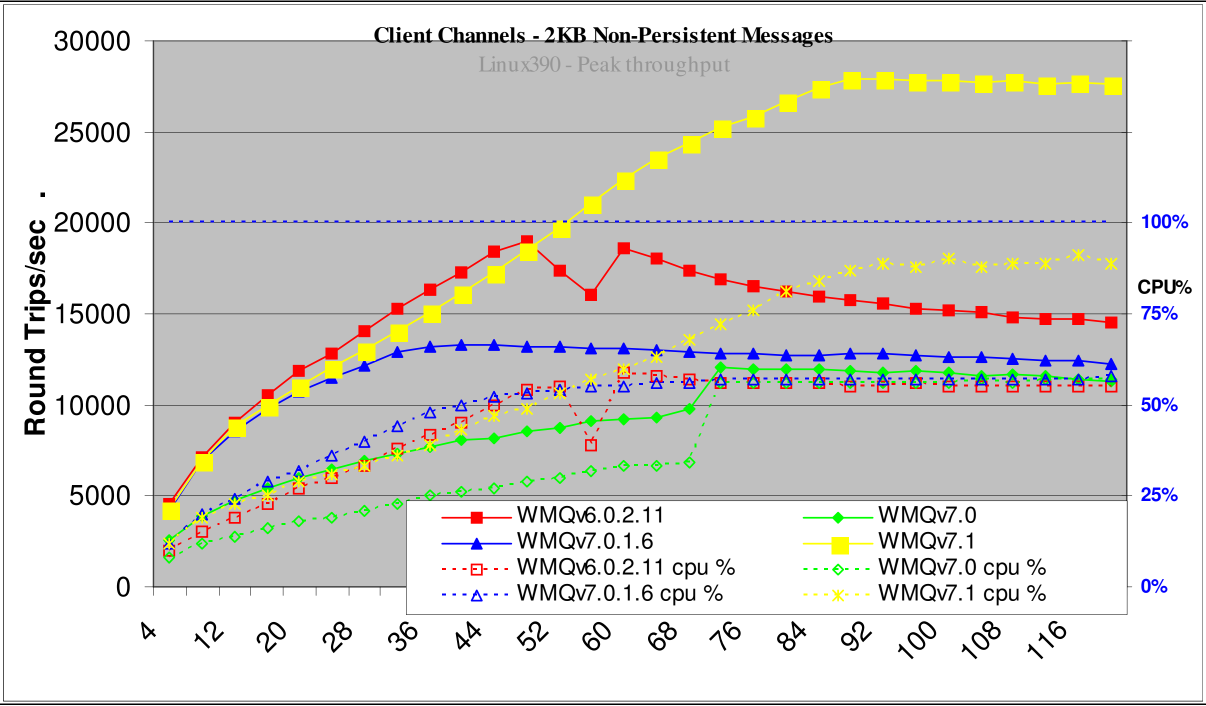


Figure 6 – Performance headline, non-persistent messages and client channels

Figure 6 and Table 4 show that the peak throughput of non-persistent messages has increased by 110% when comparing version 7.1 to 7.0.1.6 and by 47% when comparing version 7.1 to 6.0.2.11.

Test Name: Client Channels - 2KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	48	18998	0.0029	54%
WMQv7.0	72	12104	0.0072	56%
WMQv7.0.1.6	40	13284	0.0036	50%
WMQv7.1	88	27907	0.0037	87%

Table 4 – Performance headline, non-persistent messages and client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.2.2 Non-persistent Messages – Non-Trusted Client Channels

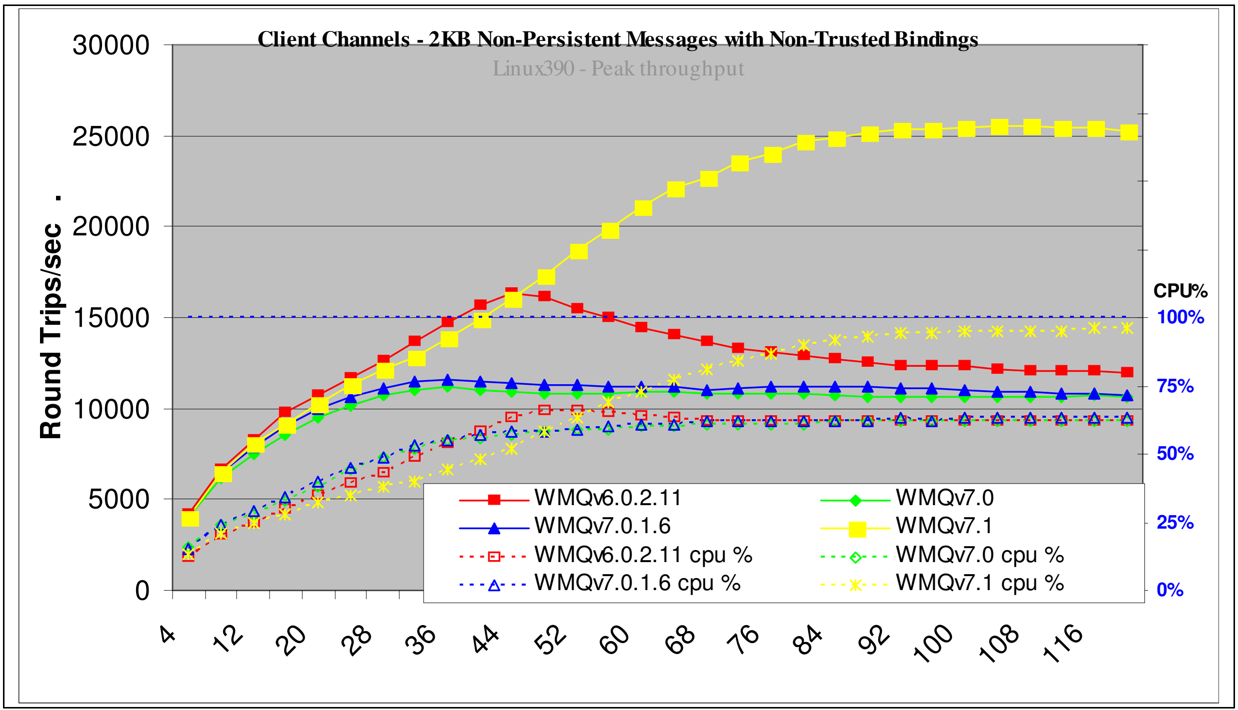


Figure 7 – Performance headline, non-persistent messages with non-trusted client channels

Figure 7 and Table 5 shows that the peak throughput of non-persistent, non-trusted messages (shared bindings - MQIBINDTYPE=NORMAL) has increased by 120% when comparing version 7.1 to 7.0.1.6 and by 56% when comparing version 7.1 to 6.0.2.11.

Test Name: Client Channels - 2KB Non-Persistent Messages with Non-Trusted Bindings	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	44	16314	0.0031	63%
WMQv7.0	36	11156	0.0035	55%
WMQv7.0.1.6	36	11583	0.0035	55%
WMQv7.1	104	25512	0.0048	95%

Table 5 – Performance headline, non-persistent messages and client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.2.3 Persistent Messages – Client Channels

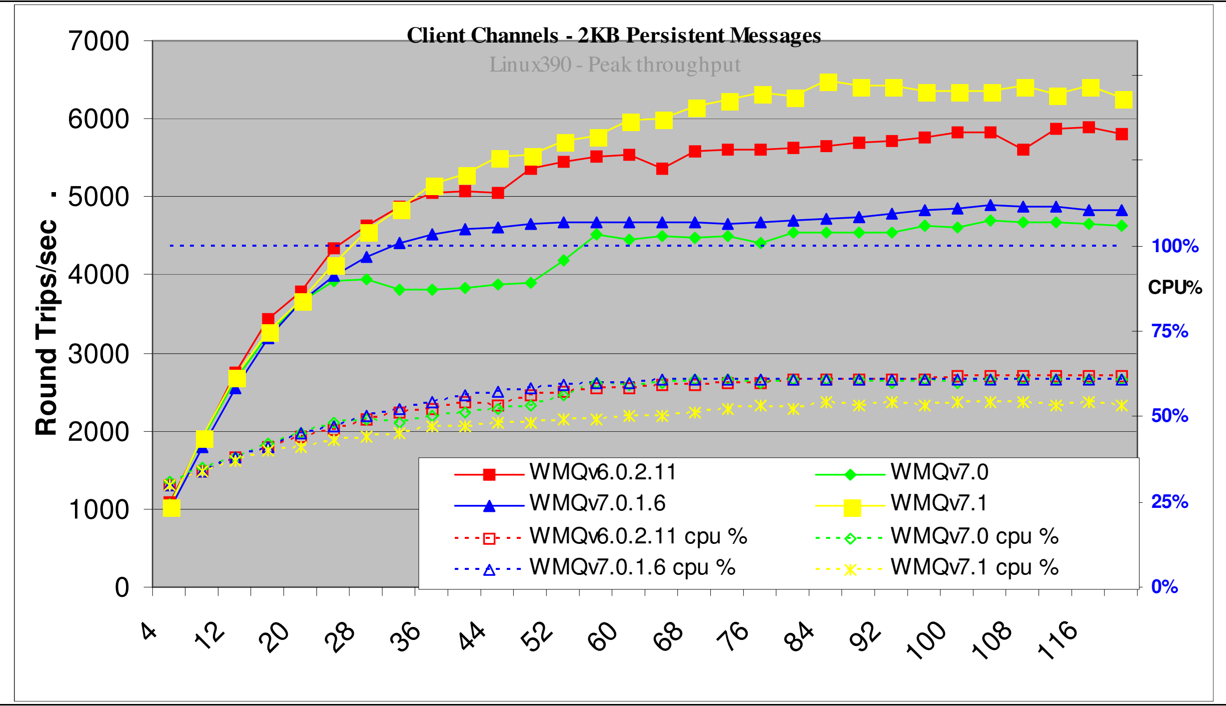


Figure 8 – Performance headline, persistent messages and client channels

Figure 8 and Table 6 shows that the peak throughput of persistent messages has increased by 33% when comparing version 7.1 to 7.0.1.6 and by 10% when comparing version 7.1 to 6.0.2.11

Test Name: Client Channels - 2KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	116	5886	0.024	62%
WMQv7.0	104	4692	0.028	61%
WMQv7.0.1.6	104	4894	0.025	61%
WMQv7.1	84	6484	0.014	54%

Table 6 – Performance headline, persistent messages and client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.2.4 Client Channels

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

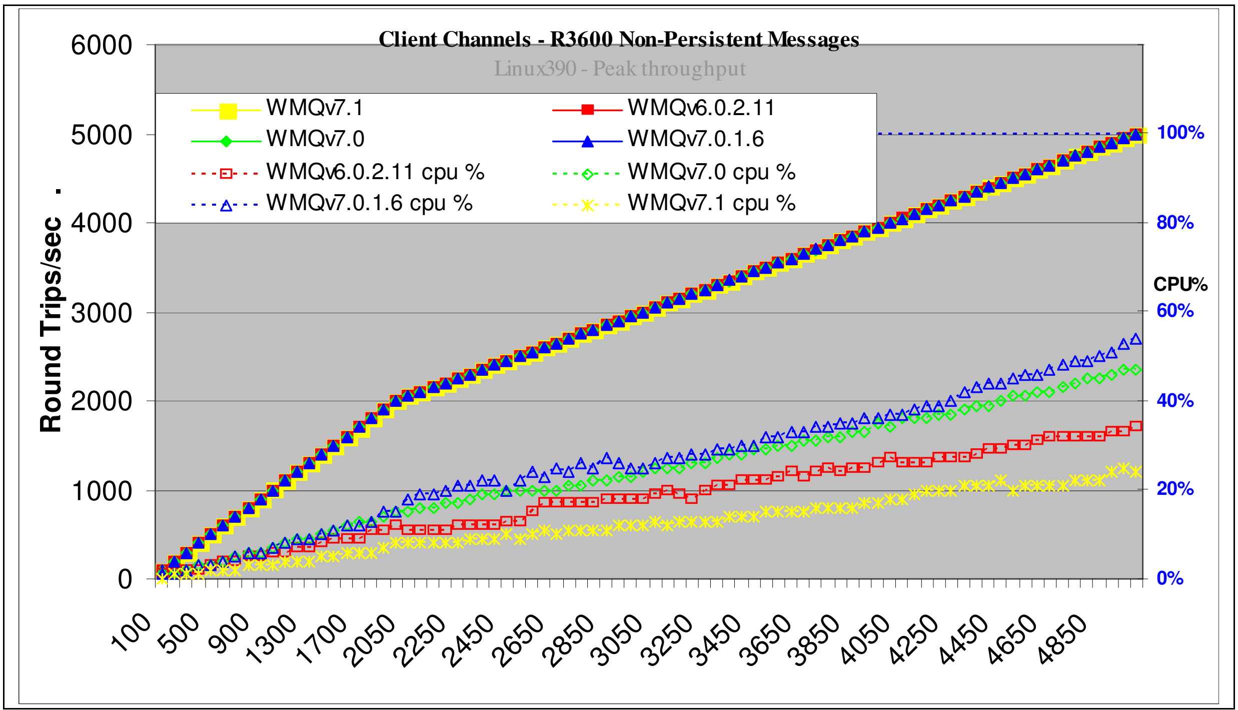


Figure 9 – 1 round trip per driving application per second, client channels and non-persistent messages

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

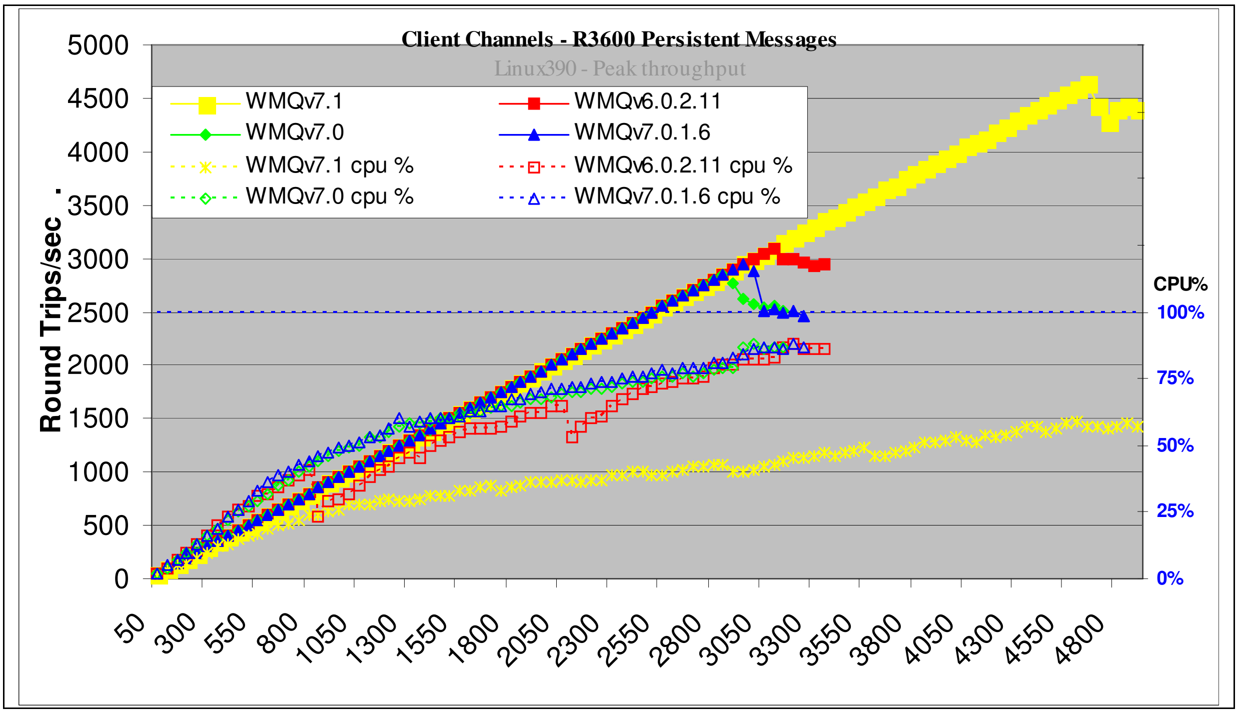


Figure 10 – 1 round trip per driving application per second, client channels, persistent messages

Figure 9, Figure 10 and Table 7 show that the peak throughput of non-persistent messages is similar when comparing version 7.1 to 7.0.1.6 and to 6.0.2.11, however version 7.1 has the lowest CPU usage. It also shows that the peak throughput of persistent messages has increased by 58% when comparing version 7.1 to 7.0.1.6 and by 50% when comparing version 7.1 to 6.0.2.11. Please note that the CPU cost is lower is 7.1 release.

Test Name: Client Channels - R3600 Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	5000	4997	0.0042	34%
WMQv7.0	5000	4998	0.0039	47%
WMQv7.0.1.6	5000	4999	0.0021	54%
WMQv7.1	5000	4998	0.0018	24%

Test Name: Client Channels - R3600 Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	3100	3098	0.0062	83%
WMQv7.0	2850	2848	0.01	79%
WMQv7.0.1.6	2950	2949	0.051	84%
WMQv7.1	4650	4649	0.19	57%

Table 7 – 1 round trip per driving application per second, client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.3 Distributed Queuing Test Scenario

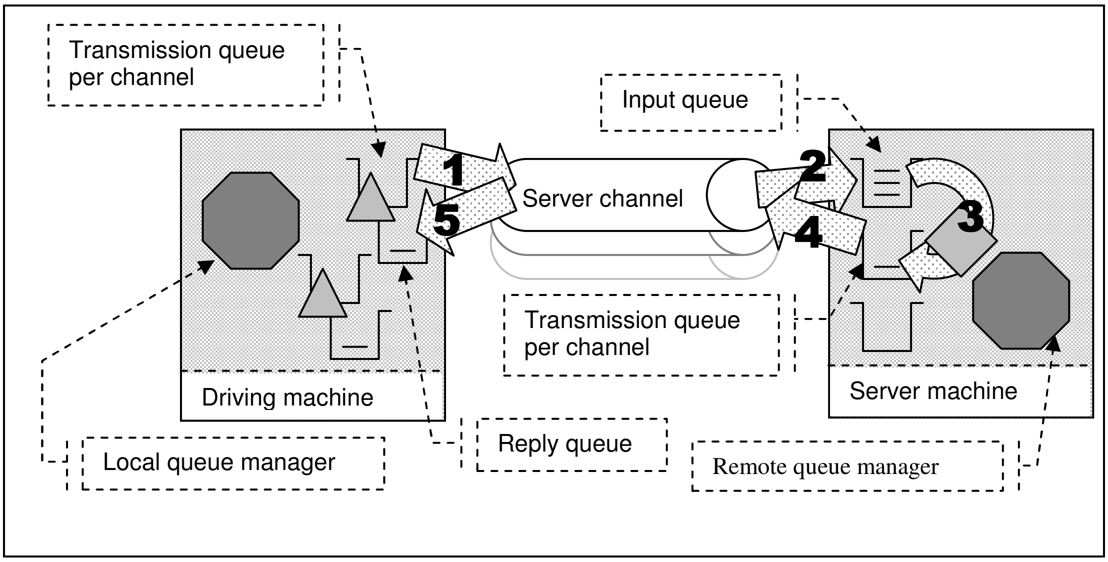


Figure 11 – Server channels between two queue managers

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

Non-persistent and persistent messages were used in the distributed queuing tests, with a message size of 2KB. 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 Non-persistent Messages – Server Channels

Figure 12, **Figure 13** and Figure 14 show the non-persistent, non-persistent non-trusted and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario (see Figure 11 on the previous page) and WebSphere MQ (versions 7.1, 7.0.1.6, 7.0 and 6.0.2.11).

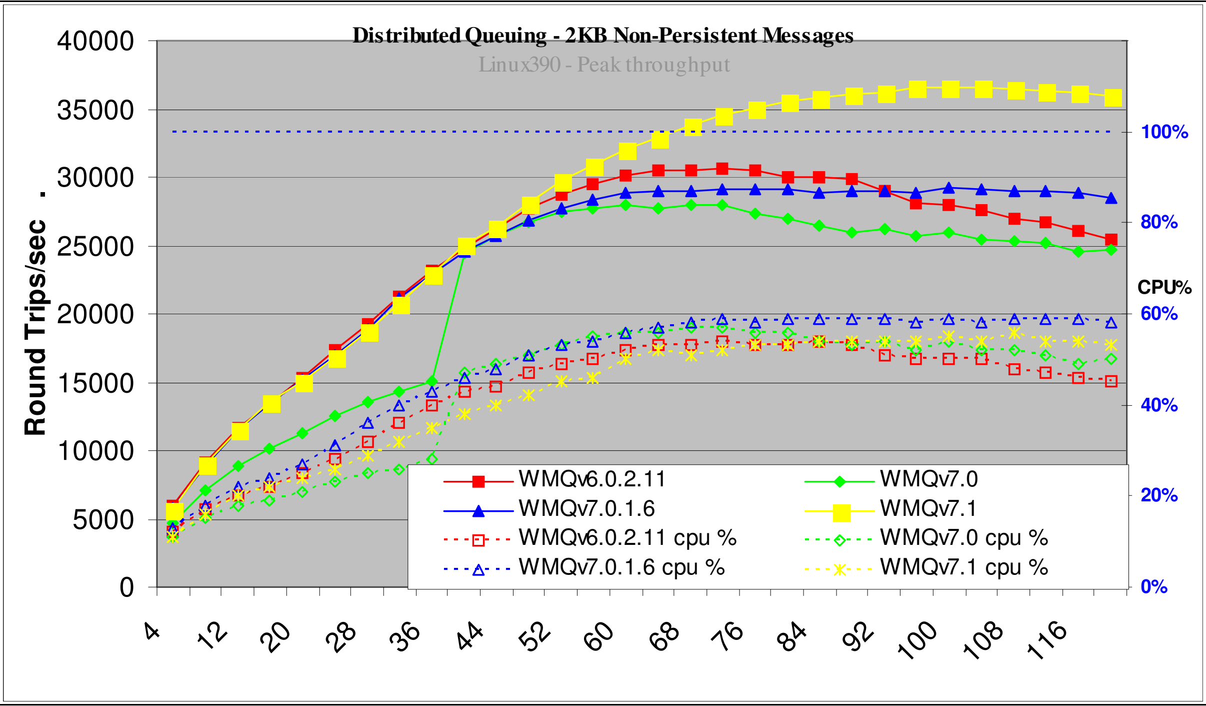


Figure 12 – Performance headline, non-persistent messages and server channels

Figure 12 and Table 8 shows that the peak throughput of non-persistent messages has increased by 25% when comparing version 7.1 to 7.0.1.6 and by 19% when comparing version 7.1 to 6.0.2.11

Test Name: Distributed Queuing - 2KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	72	30680	0.0029	54%
WMQv7.0	60	27991	0.0026	56%
WMQv7.0.1.6	100	29202	0.0039	59%
WMQv7.1	104	36545	0.0033	54%

Table 8 – Performance headline, non-persistent messages and server channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.3.2 Non-Persistent non-Trusted – Server Channels

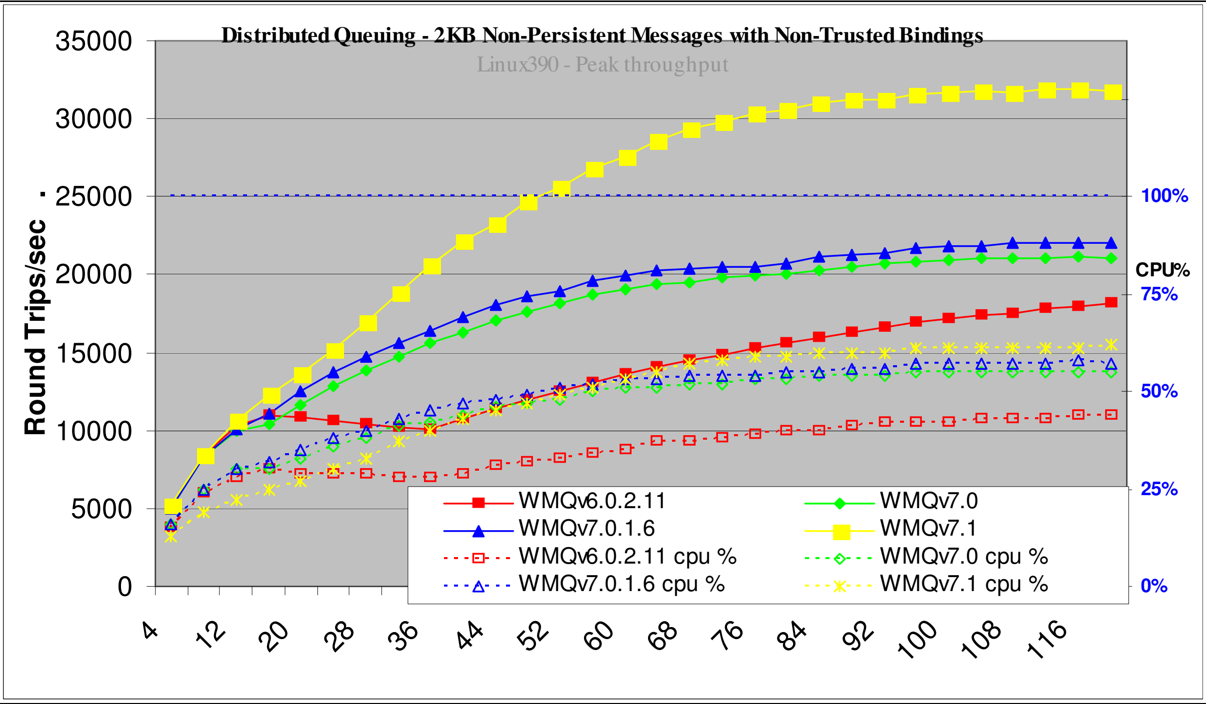


Figure 13 – Performance headline, non-persistent, not trusted messages and server channels

Figure 13 and Table 9 shows that the peak throughput Table 9 of non-persistent, non-trusted messages has increased by 45% when comparing version 7.1 to 7.0.1.6 and by 76% when comparing version 7.1 to 6.0.2.11.

Test Name: Distributed Queuing - 2KB Non-Persistent Messages with Non-Trusted Bindings	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	120	18156	0.0078	44%
WMQv7.0	116	21127	0.0067	55%
WMQv7.0.1.6	116	22059	0.006	58%
WMQv7.1	116	31883	0.0043	61%

Table 9 – Performance headline, non-persistent, non trusted messages and server channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

2.3.3 Persistent Messages – Server Channels

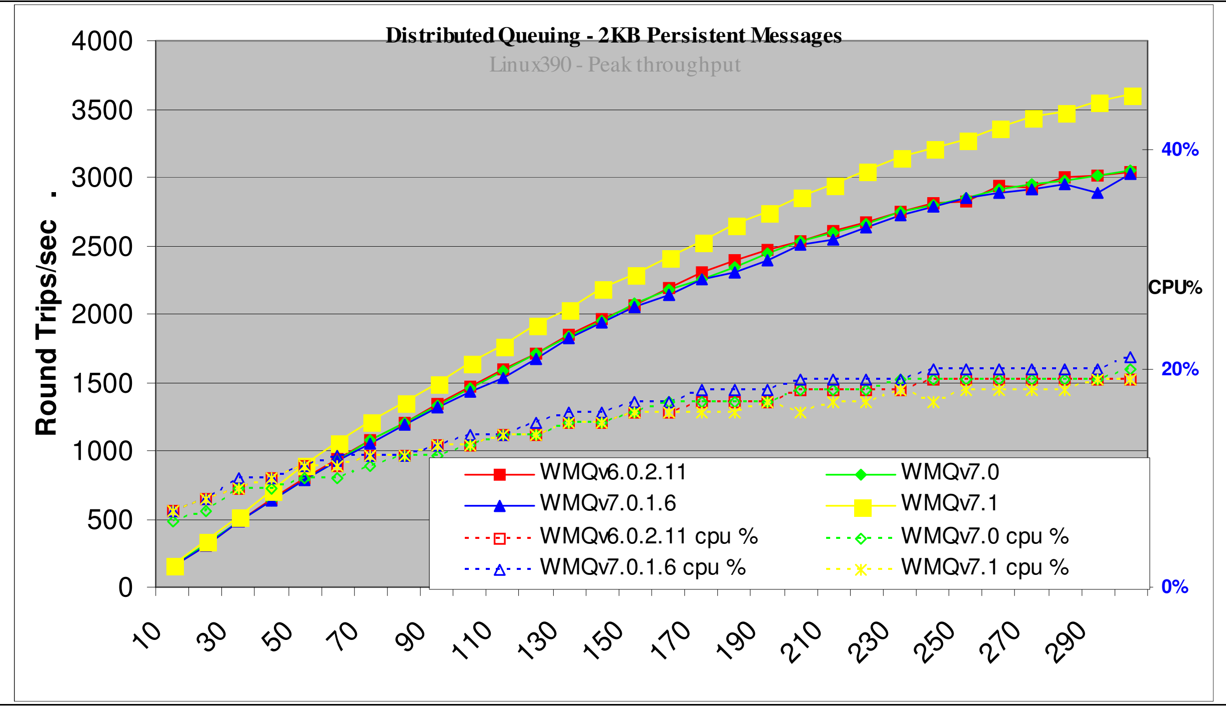


Figure 14 – Performance headline, persistent messages and server channels

Figure 14 and Table 10 shows that the peak throughput of persistent messages has increased by 19% when comparing version 7.1 to 7.0.1.6 and by 18% when comparing version 7.1 to 6.0.2.11.

Test Name: Distributed Queuing - 2KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	300	3041	0.11	19%
WMQv7.0	300	3046	0.12	20%
WMQv7.0.1.6	300	3020	0.12	21%
WMQv7.1	300	3603	0.092	19%

Table 10 – Performance headline, persistent messages and server channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

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 non-persistent messaging tests, and 2 pairs for the persistent message tests.

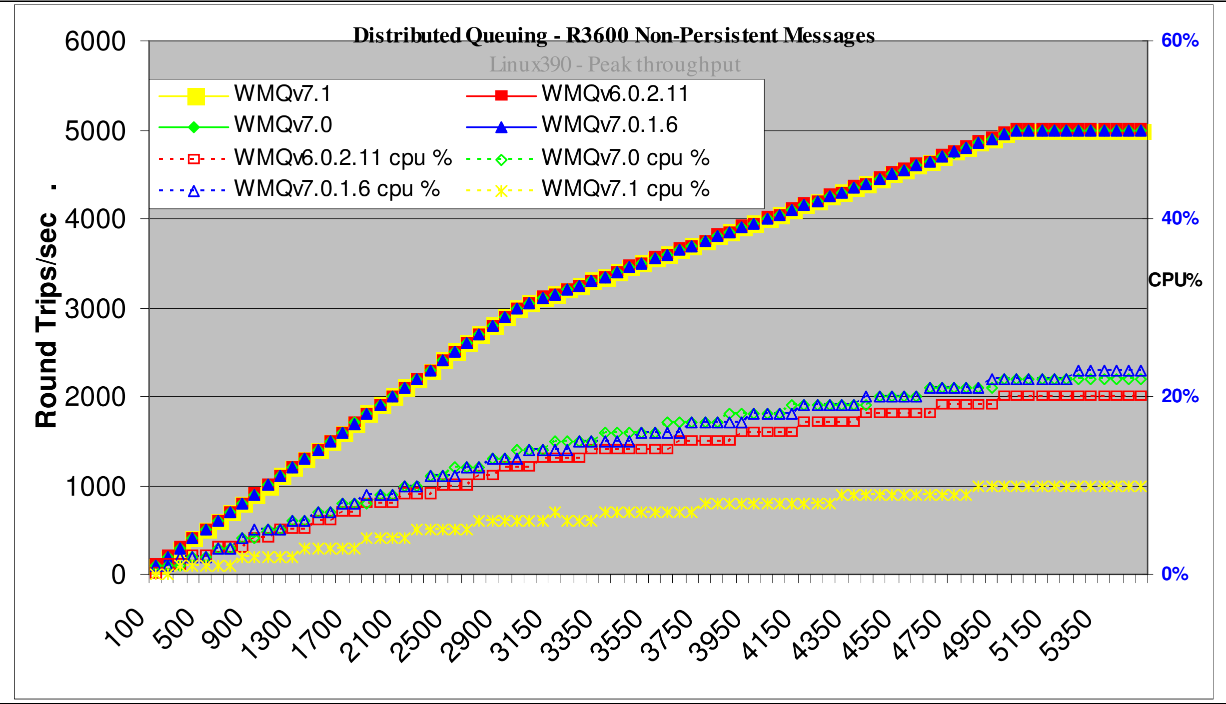


Figure 15 – 1 round trip per driving application per second, server channel, non-persistent messages

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

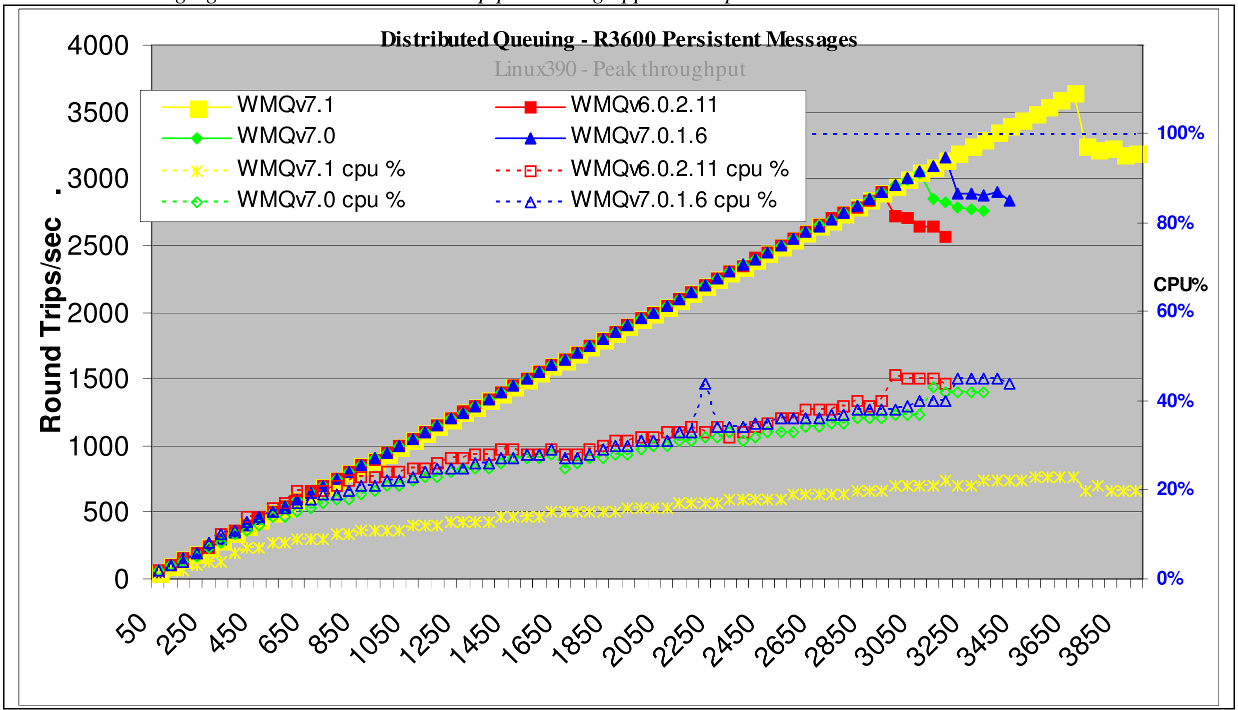


Figure 16 – 1 round trip per driving application per second, server channel, persistent messages

Figure 15, Figure 16 and Table 11 shows that the peak throughput of non-persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11 and for persistent messages has increased by 16% when comparing version 7.1 to 7.0.1.6 and by 26% when comparing version 7.1 to 6.0.2.11. CPU costs are lower in version 7.1 than other releases.

Test Name: Distributed Queuing - R3600 Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	5000	4998	0.001	20%
WMQv7.0	5500	4998	0.00094	22%
WMQv7.0.1.6	5200	4998	0.001	22%
WMQv7.1	5400	4998	0.00092	10%

Test Name: Distributed Queuing - R3600 Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	2900	2898	0.14	40%
WMQv7.0	3050	3048	0.15	37%
WMQv7.0.1.6	3150	3152	0.17	40%
WMQv7.1	3650	3648	0.09	23%

Table 11 – 1 round trip per driving application per second, client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3 Large Messages

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

3.1.1 50bytes to 32KB

Figure 17 show that the response time for MQPut/MQGet for non-persistent message sizes between 50bytes and 32KB.

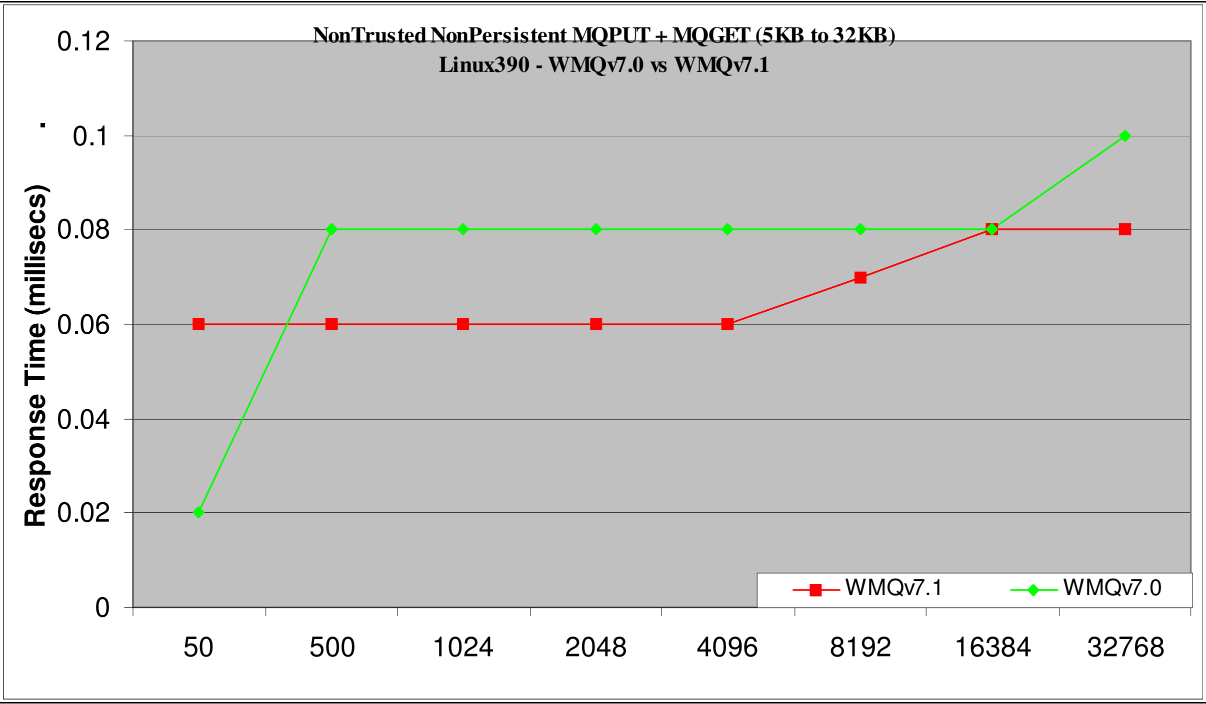


Figure 17 –The effect of non-persistent message size on MQI response time (50byte - 32KB)

Figure 18 show that the response for MQPut/MQGet pairs for persistent message sizes between 50bytes and 32KB.

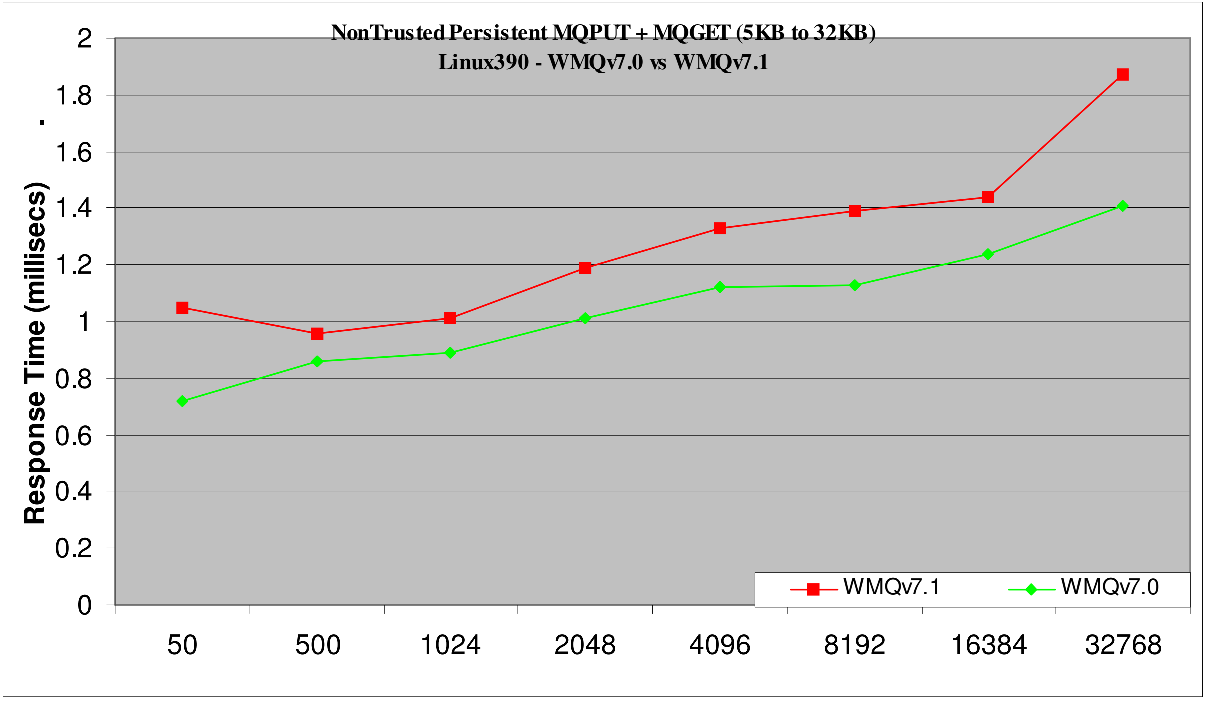


Figure 18 –The effect of persistent message size on MQI response time (50byte - 32KB)

3.1.2 32KB to 2MB

Figure 19 show that the response time for MQPut/MQGet pairs has improved for all non-persistent message sizes between 32KB and 2MB.

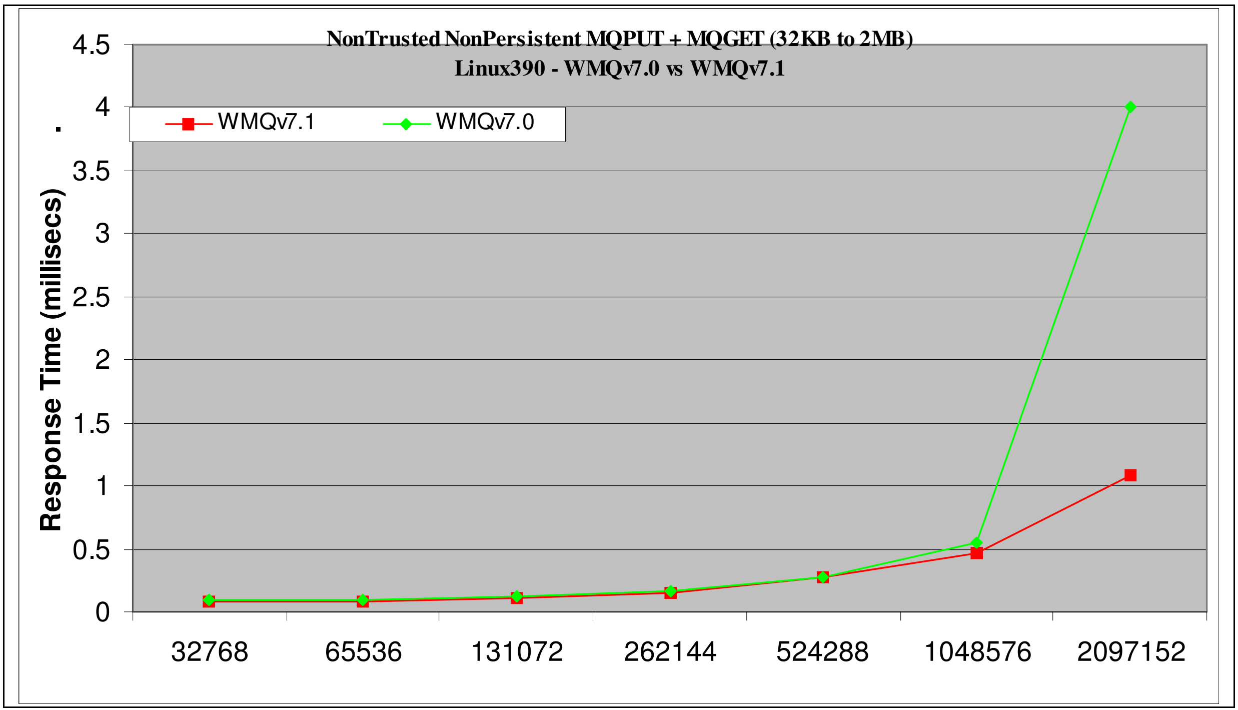


Figure 19 –The effect of non-persistent message size on MQI response time (32KB – 2MB)

Figure 20 show that the response for MQPut/MQGet pairs for persistent message sizes between 32KB and 2MB.

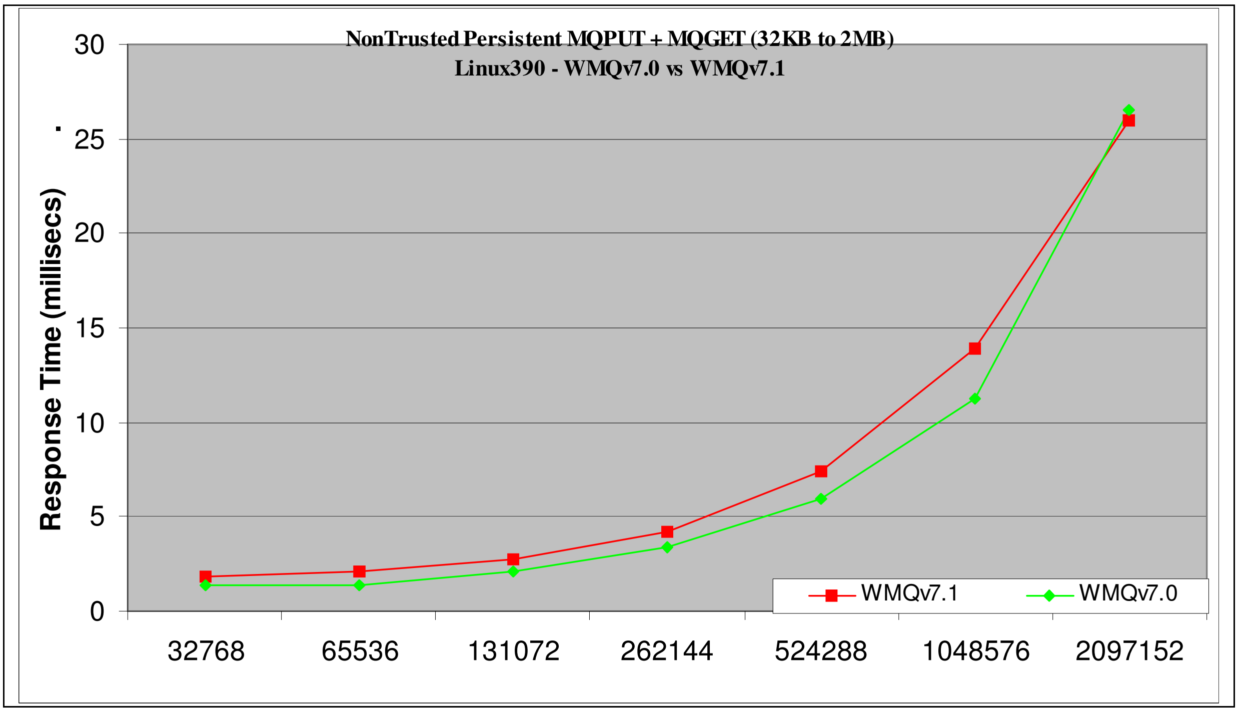


Figure 20 –The effect of persistent message size on MQI response time (32KB – 2MB)

3.1.3 2MB to 100MB

Figure 21 Response time for MQPut/MQGet pairs for NP message between 2MB and 100MB.

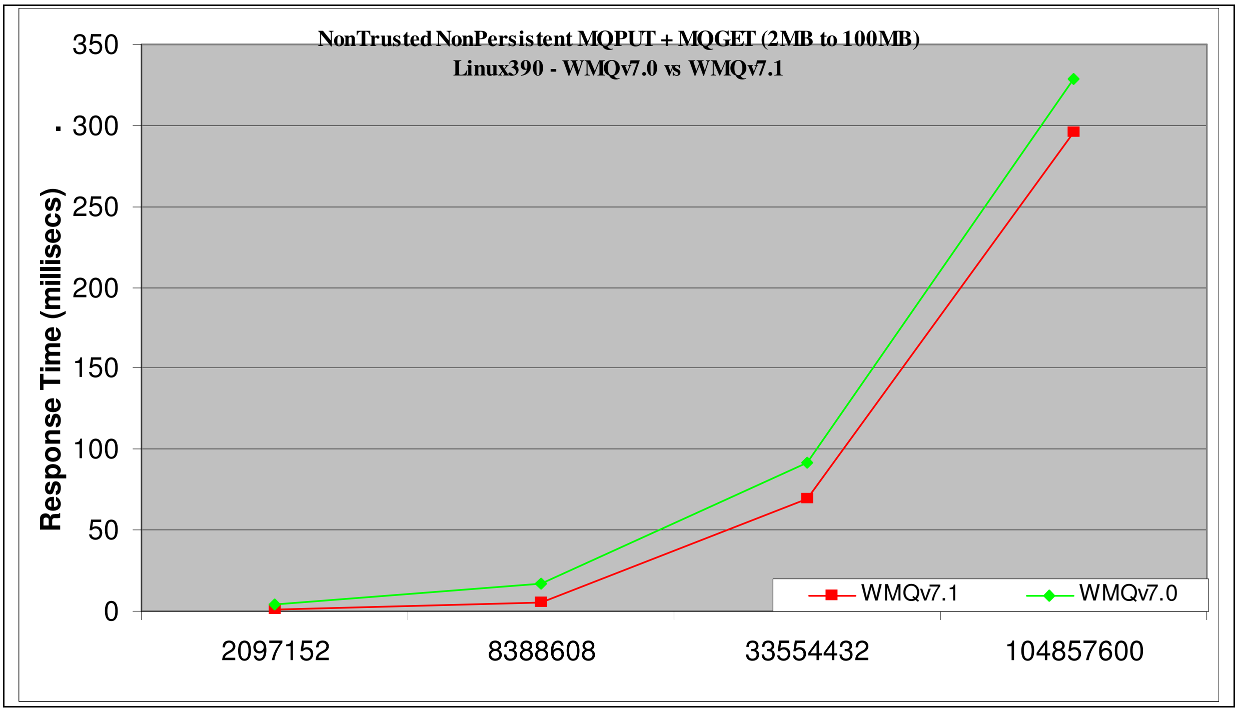


Figure 21 –The effect of non-persistent message size on MQI response time (2MB – 100MB)

Figure 22 The response for MQPut/MQGet pairs for persistent message sizes between 2MB and 32MB.

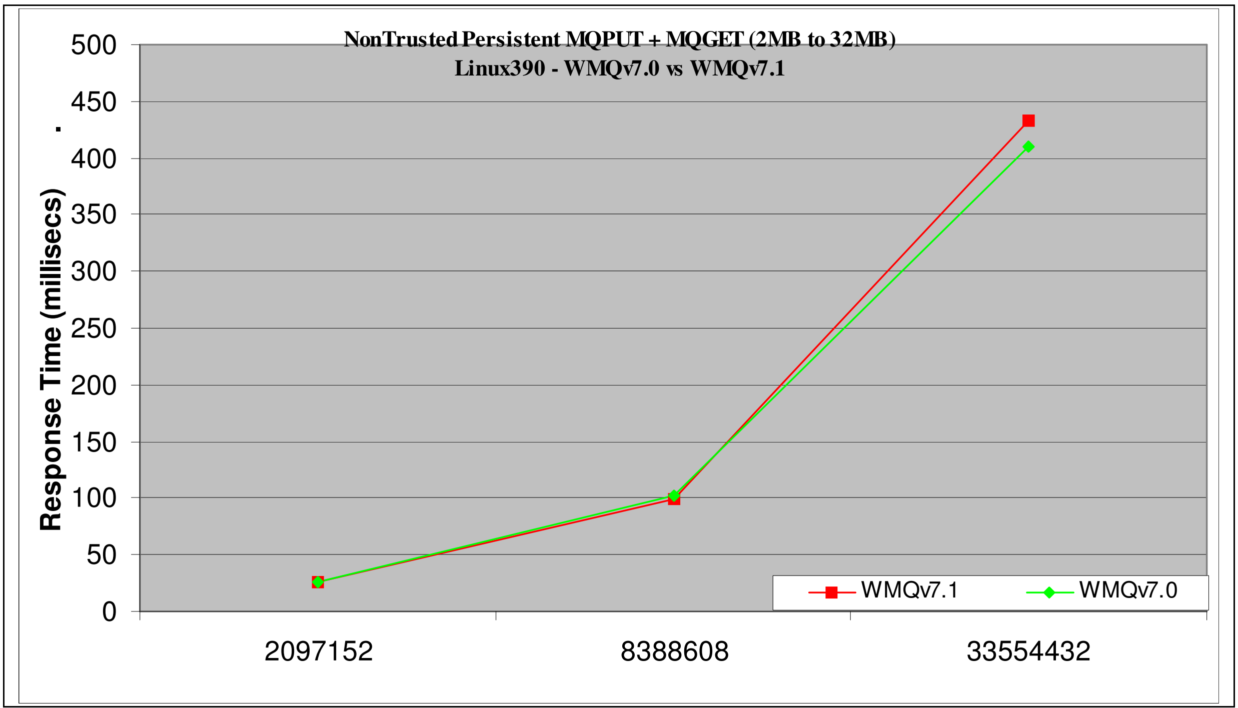


Figure 22 –The effect of persistent message size on MQI response time (2MB – 32MB)

3.2 20KB Messages

3.2.1 Local Queue Manager

Figure 23 and Figure 24 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

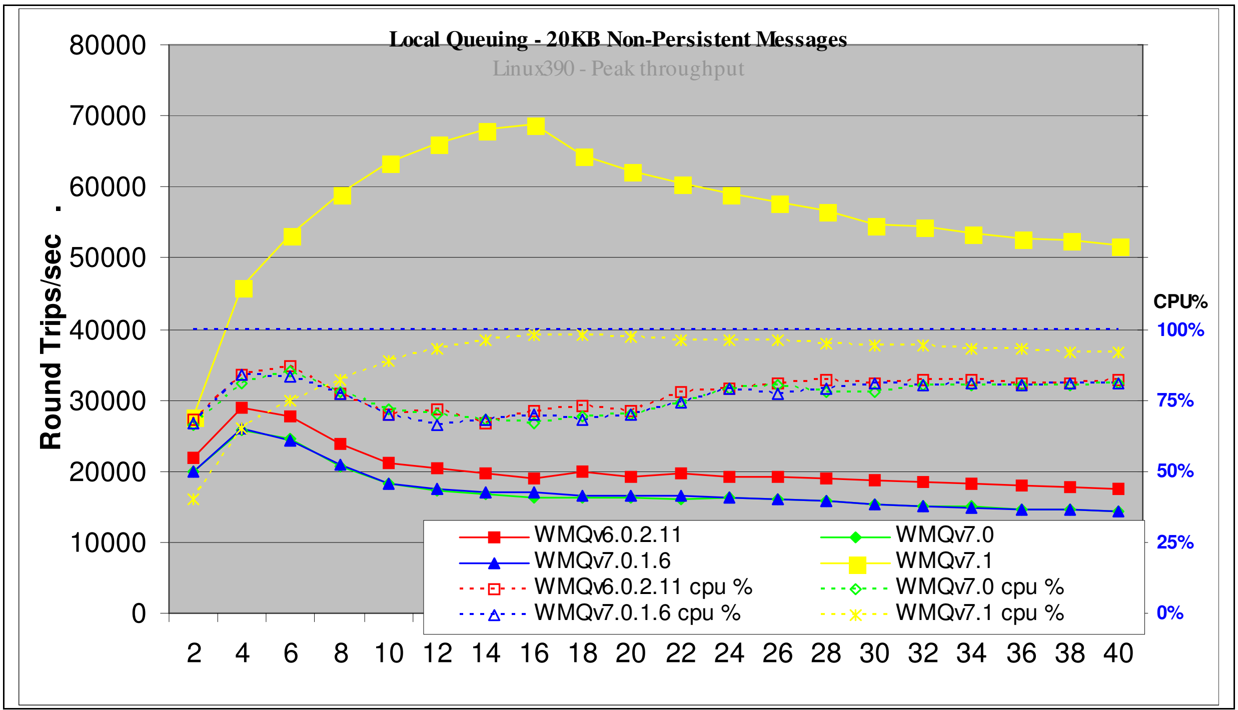


Figure 23 – 20KB non-persistent messages, local queue manager

Figure 23 and Table 12 shows that the peak throughput of non-persistent messages has increased by 164% when comparing version 7.1 to 7.0.1.6 and by 139% when comparing version 7.1 to 6.0.2.11.

Test Name: Local Queuing - 20KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	4	28845	0.00019	84%
WMQv7.0	4	25837	0.00013	81%
WMQv7.0.1.6	4	26098	0.00032	84%
WMQv7.1	16	68806	0.0002	98%

Table 12 – 20KB non-persistent messages, local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.2.1.1 Persistent Messages

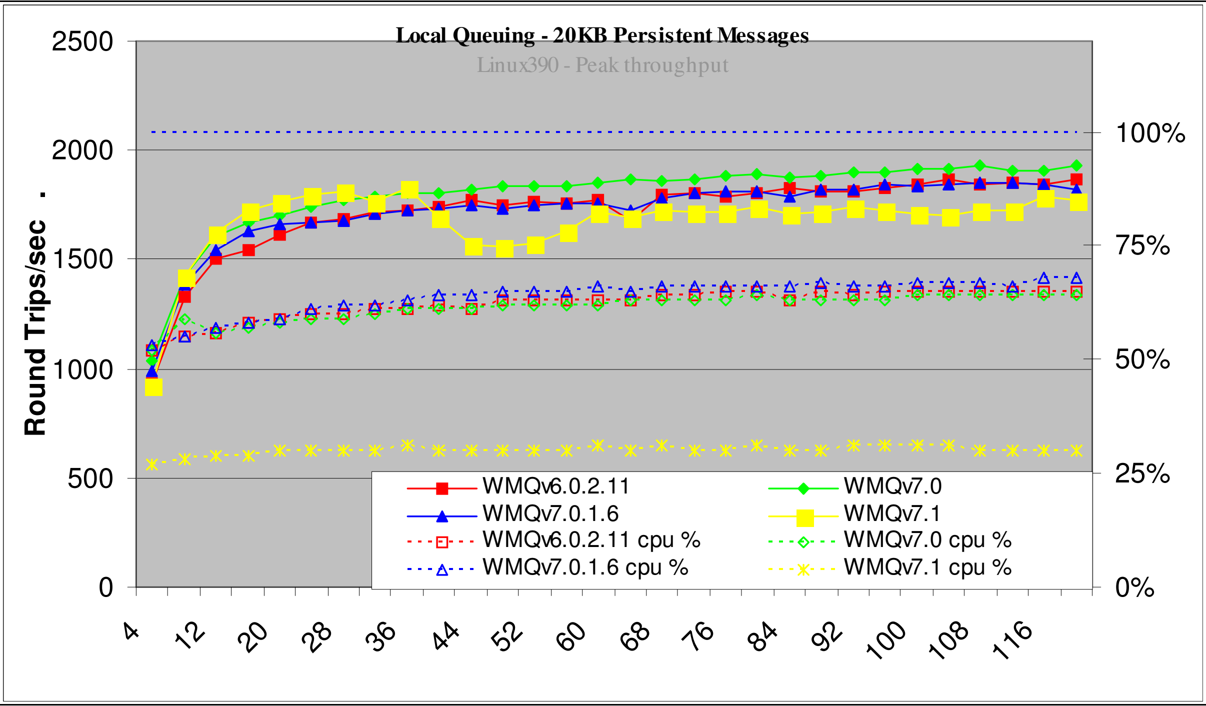


Figure 24 – 20KB persistent messages, local queue manager

Figure 24 and Table 13 shows that the peak throughput of persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11, however at certain application numbers version 7.1 shows degradations against previous releases.

Test Name: Local Queuing - 20KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	120	1868	0.076	65%
WMQv7.0	120	1932	0.073	64%
WMQv7.0.1.6	108	1848	0.084	67%
WMQv7.1	36	1824	0.022	31%

Table 13 – 20KB persistent messages, local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.2.2 Client Channel

Figure 25 and Figure 26 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

3.2.2.1 Non-persistent Messages

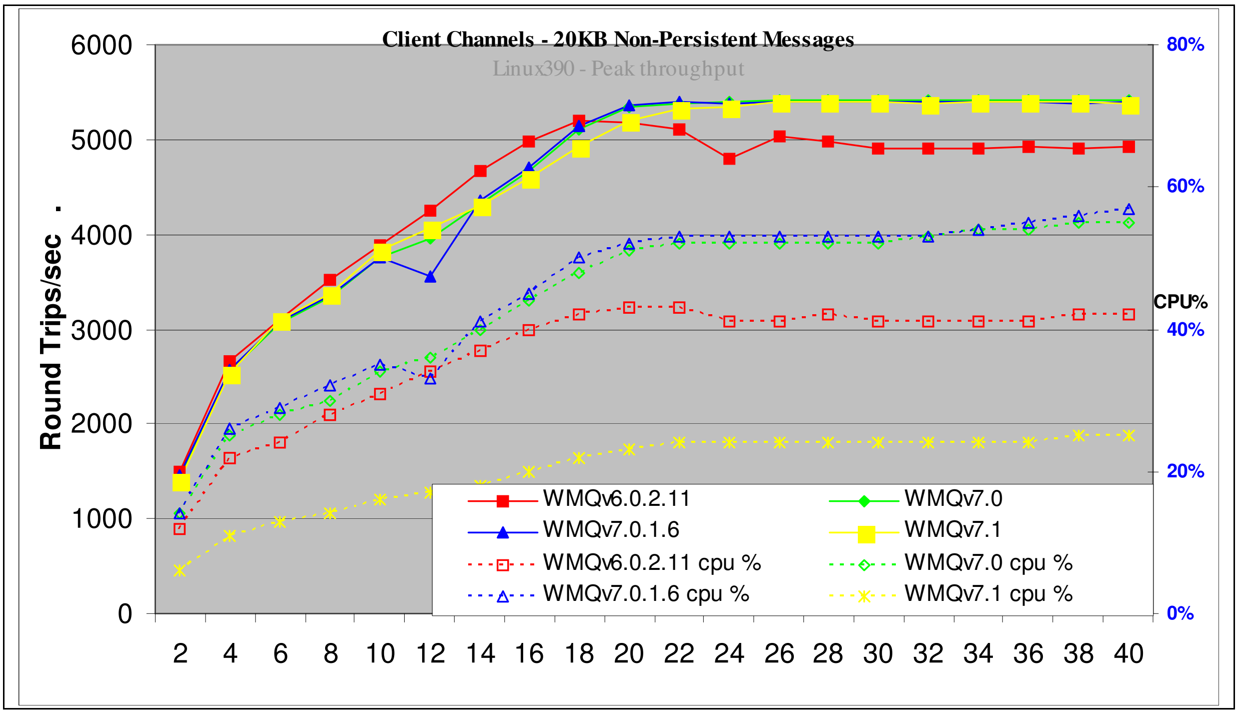


Figure 25 – 20KB non-persistent messages, client channels

Figure 25 and Table 14 shows that the peak throughput of non-persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11.

Test Name: Client Channels - 20KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	18	5199	0.0041	42%
WMQv7.0	36	5415	0.0078	54%
WMQv7.0.1.6	34	5395	0.0074	54%
WMQv7.1	38	5404	0.0083	25%

Table 14 – 20KB non-persistent messages, client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.2.2.2 Persistent Messages

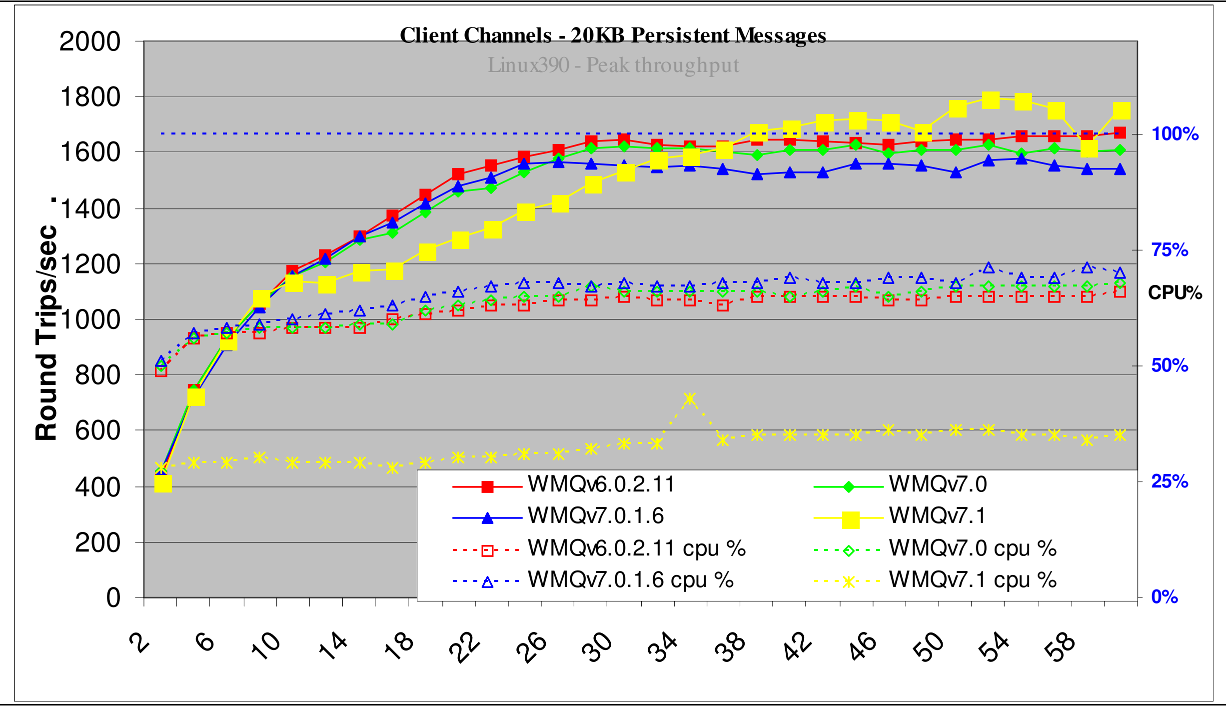


Figure 26 – 20KB persistent messages, client channels

Figure 26 and Table 15 shows that the peak throughput of persistent messages has increased by 14% when comparing version 7.1 to 7.0.1.6 and by 7% when comparing version 7.1 to 6.0.2.11. Locking improvements in WMQv7.1 have improved the right hand side of the graphs but came with path length costs that may affect the rate of growth on left hand side of the graph when there is only a small number of parallel applications.

Test Name: Client Channels - 20KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	60	1674	0.041	66%
WMQv7.0	44	1630	0.031	67%
WMQv7.0.1.6	54	1577	0.04	69%
WMQv7.1	52	1794	0.033	36%

Table 15 – 20KB persistent messages, client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.2.3 Distributed Queuing

Figure 27 and Figure 28 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario.

3.2.3.1 Non-persistent Messages

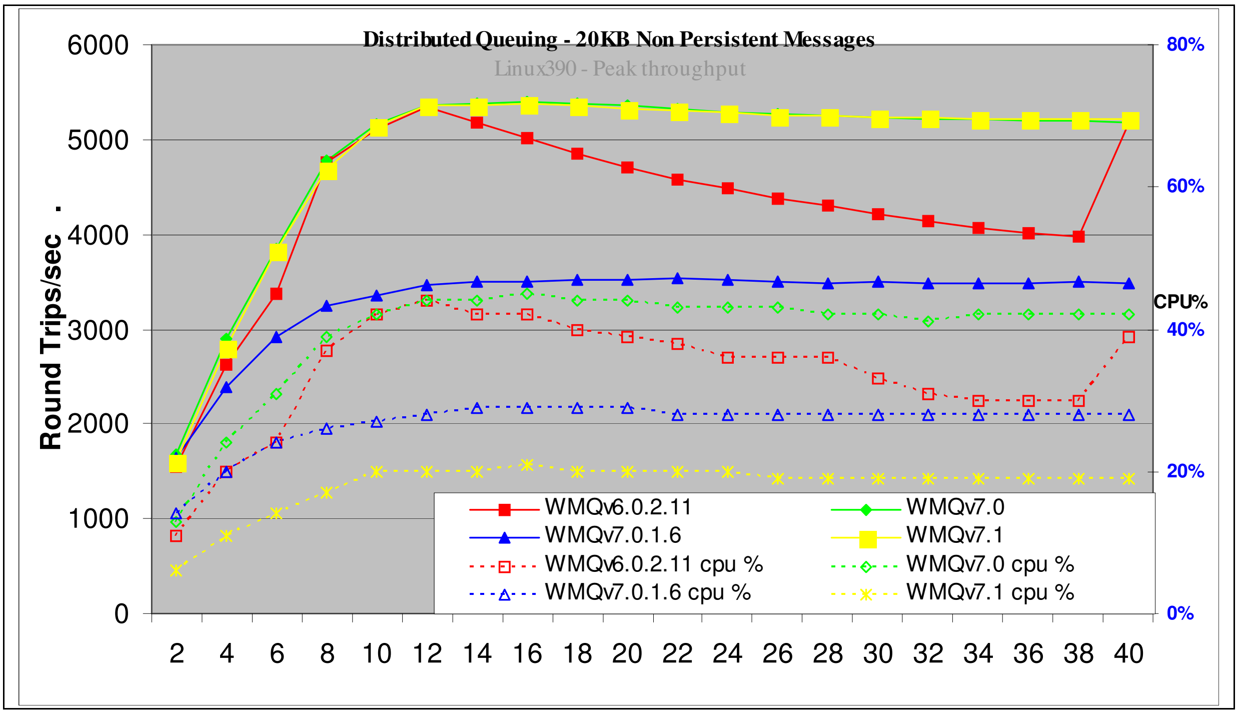


Figure 27 – 20KB non-persistent messages, distributed queuing

Figure 27 and Table 16 shows that the peak throughput of non-persistent messages has increased by 52% when comparing version 7.1 to 7.0.1.6 and is unchanged when comparing version 7.1 to 6.0.2.11.

Test Name: Distributed Queuing - 20KB Non Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	12	5340	0.0026	44%
WMQv7.0	16	5399	0.0034	45%
WMQv7.0.1.6	22	3536	0.0072	28%
WMQv7.1	16	5377	0.0035	21%

Table 16 – 20KB non-persistent messages, client channels

Note: *The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time*

3.2.3.2 Persistent Messages

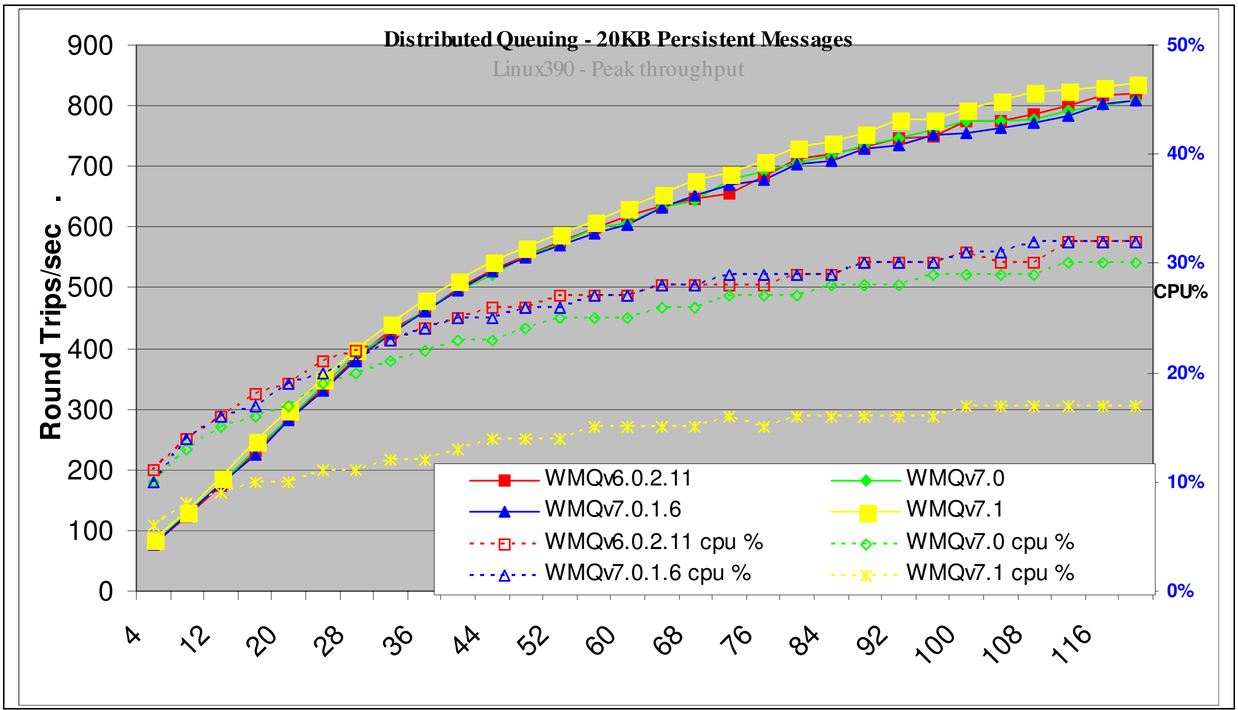


Figure 28 – 20KB persistent messages, distributed queuing

Figure 28 and Table 17 shows that the peak throughput of persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11.

Test Name: Distributed Queuing - 20KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	120	820	0.16	32%
WMQv7.0	120	810	0.17	30%
WMQv7.0.1.6	120	809	0.16	32%
WMQv7.1	120	838	0.18	17%

Table 17 – 20KB persistent messages, client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.3 200K Messages

3.3.1 Local Queue Manager

Figure 29 and Figure 30 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

3.3.1.1 Non-persistent Messages

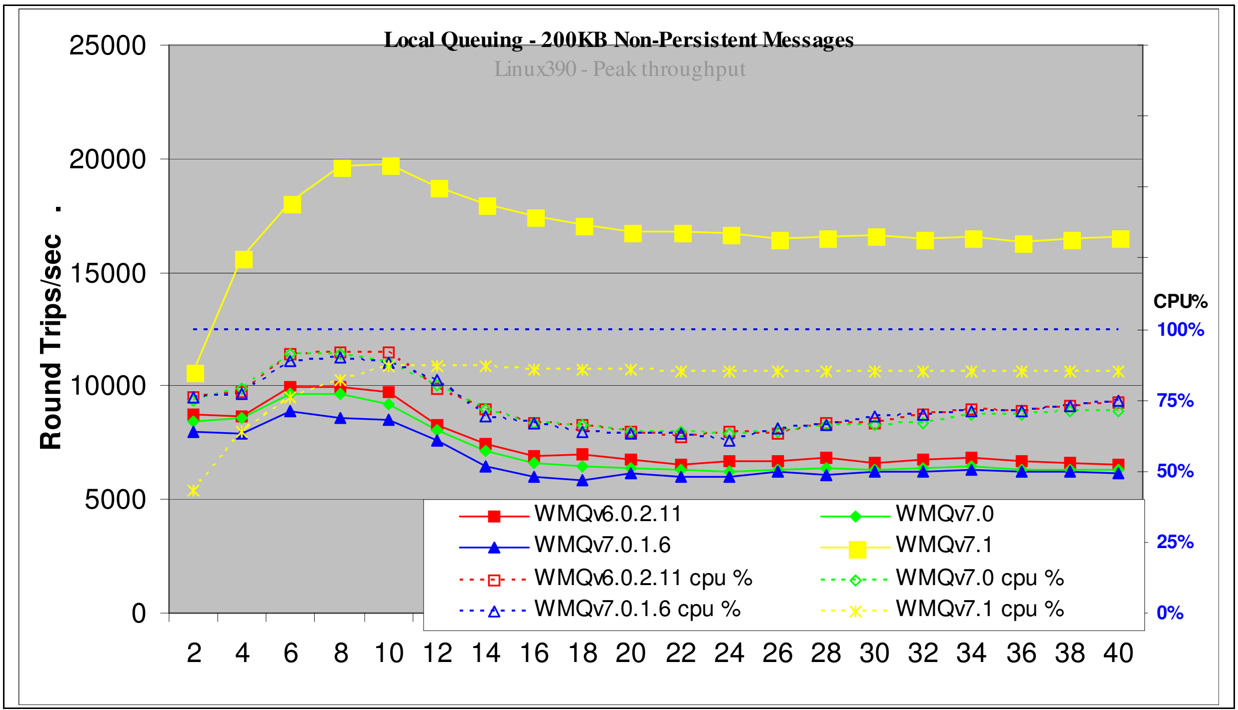


Figure 29 – 200KB non-persistent messages, local queue manager

Figure 29 and Table 18 shows that the peak throughput of non-persistent messages has increased by 122% when comparing version 7.1 to 7.0.1.6 and by 98% when comparing version 7.1 to 6.0.2.11.

Test Name: Local Queuing - 200KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	8	9981	0.001	92%
WMQv7.0	6	9686	0.00076	91%
WMQv7.0.1.6	6	8896	0.00096	89%
WMQv7.1	10	19728	0.00051	87%

Table 18 – 200KB non-persistent messages, local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.3.1.2 Persistent Messages

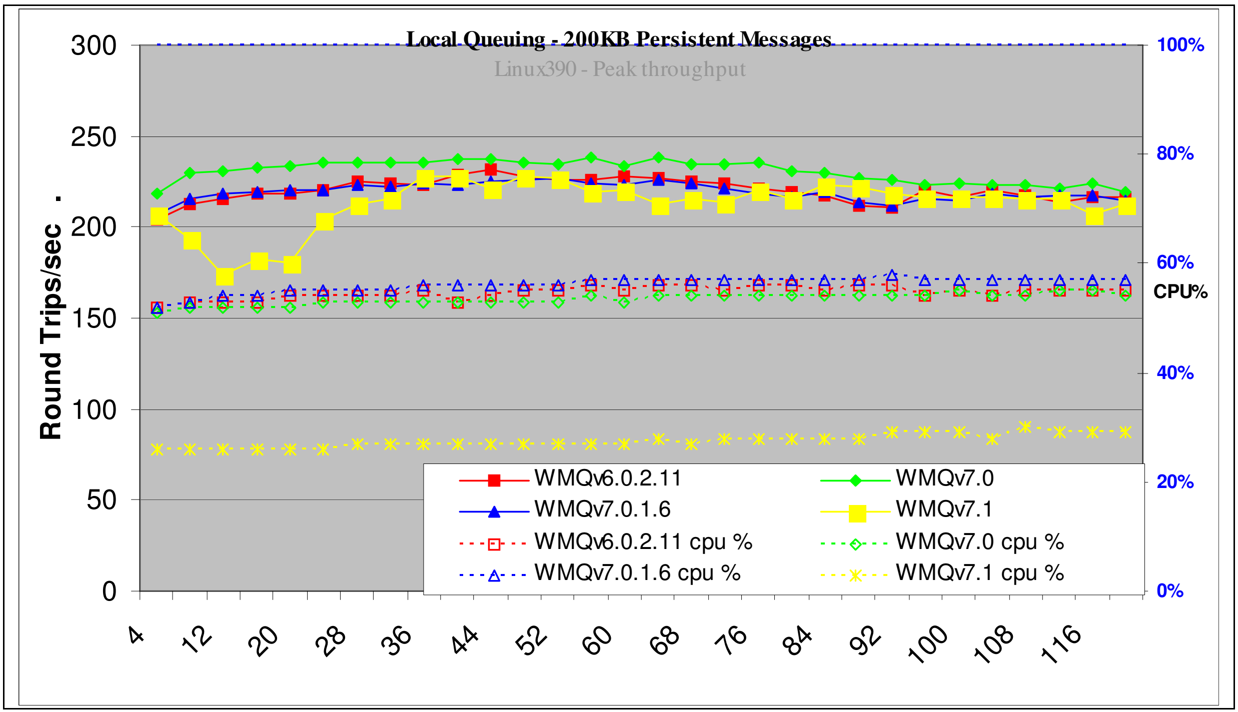


Figure 30 – 200KB persistent messages, local queue manager

Figure 30 and Table 19 shows that the peak throughput of persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11.

Test Name: Local Queuing - 200KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	44	232	0.22	54%
WMQv7.0	56	238	0.27	54%
WMQv7.0.1.6	52	227	0.26	56%
WMQv7.1	40	228	0.2	27%

Table 19 – 200KB persistent messages, local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.3.2 Client Channel

Figure 31 and Figure 32 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

3.3.2.1 Non-persistent Messages

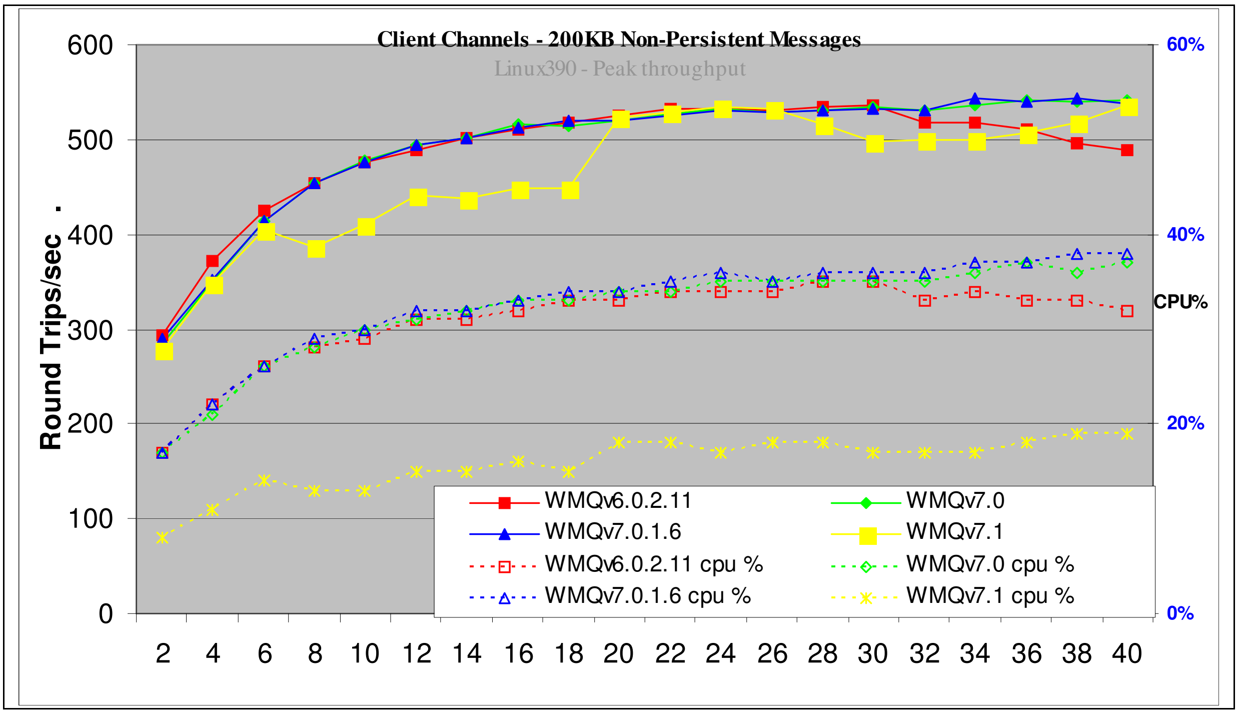


Figure 31 – 200KB non-persistent messages, client channels

Figure 31 and Table 20 shows that the peak throughput of non-persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11.

Test Name: Client Channels - 200KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	30	536	0.066	35%
WMQv7.0	40	541	0.086	37%
WMQv7.0.1.6	38	544	0.083	38%
WMQv7.1	40	536	0.088	19%

Table 20 – 200KB non-persistent messages, client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.3.2.2 Persistent Messages

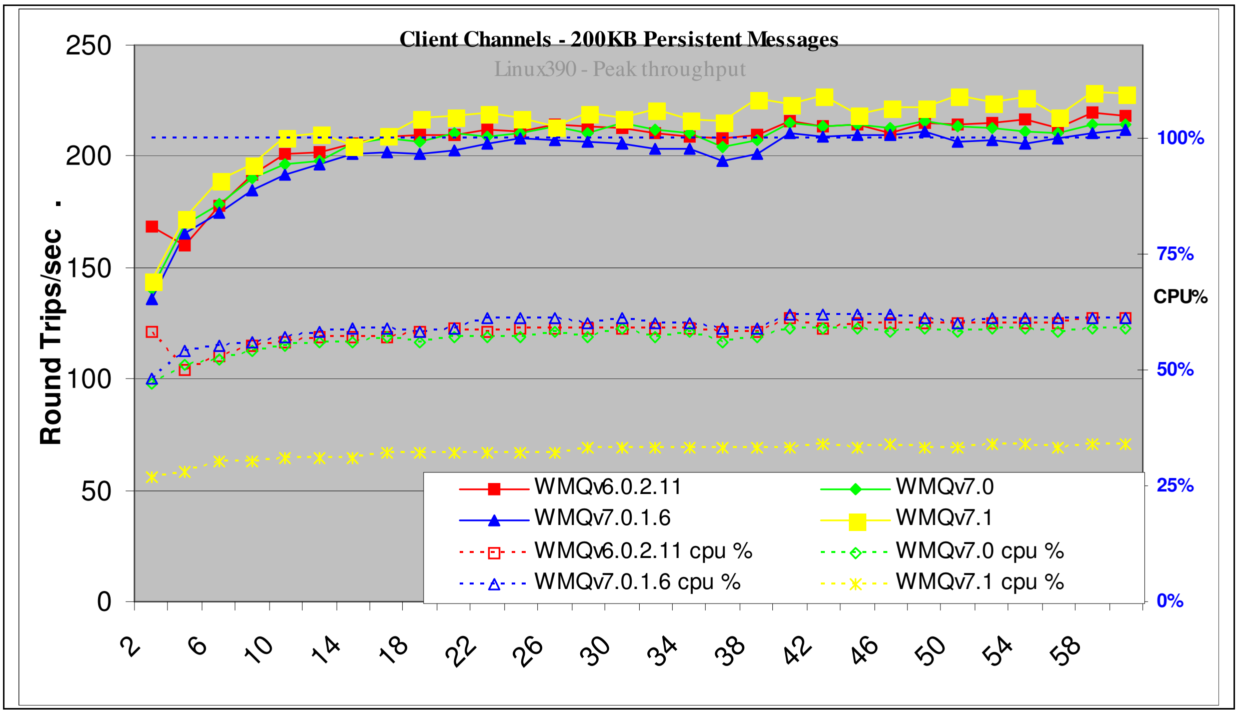


Figure 32 – 200KB persistent messages, client channels

Figure 32 and Table 21 shows that the peak throughput of persistent messages has increased by 8% when comparing version 7.1 to 7.0.1.6 and is similar when comparing version 7.1 to 6.0.2.11.

Test Name: Client Channels - 200KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	58	220	0.3	61%
WMQv7.0	48	216	0.26	59%
WMQv7.0.1.6	60	212	0.33	61%
WMQv7.1	58	229	0.3	34%

Table 21 – 200KB persistent messages, client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.3.3 Distributed Queuing

Figure 33 and Figure 34 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario

3.3.3.1 Non-persistent Messages

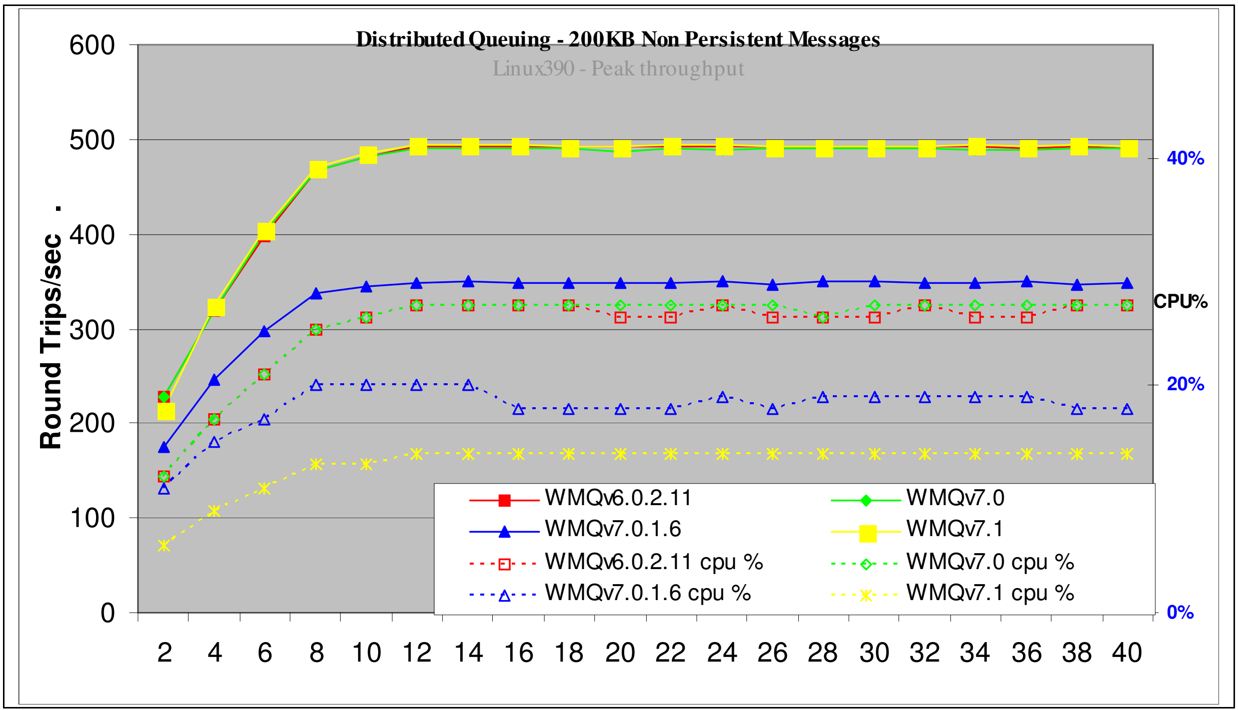


Figure 33 – 200KB non-persistent messages, distributed queuing

Figure 33 and Table 22 shows that the peak throughput of non-persistent messages has increased by 41% when comparing version 7.1 to 7.0.1.6 and is similar when comparing version 7.1 to 6.0.2.11.

Test Name: Distributed Queuing - 200KB Non Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	12	493	0.028	27%
WMQv7.0	12	491	0.028	27%
WMQv7.0.1.6	14	350	0.046	20%
WMQv7.1	34	495	0.081	14%

Table 22 – 200KB non-persistent messages, distributed queuing

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.3.3.2 Persistent Messages

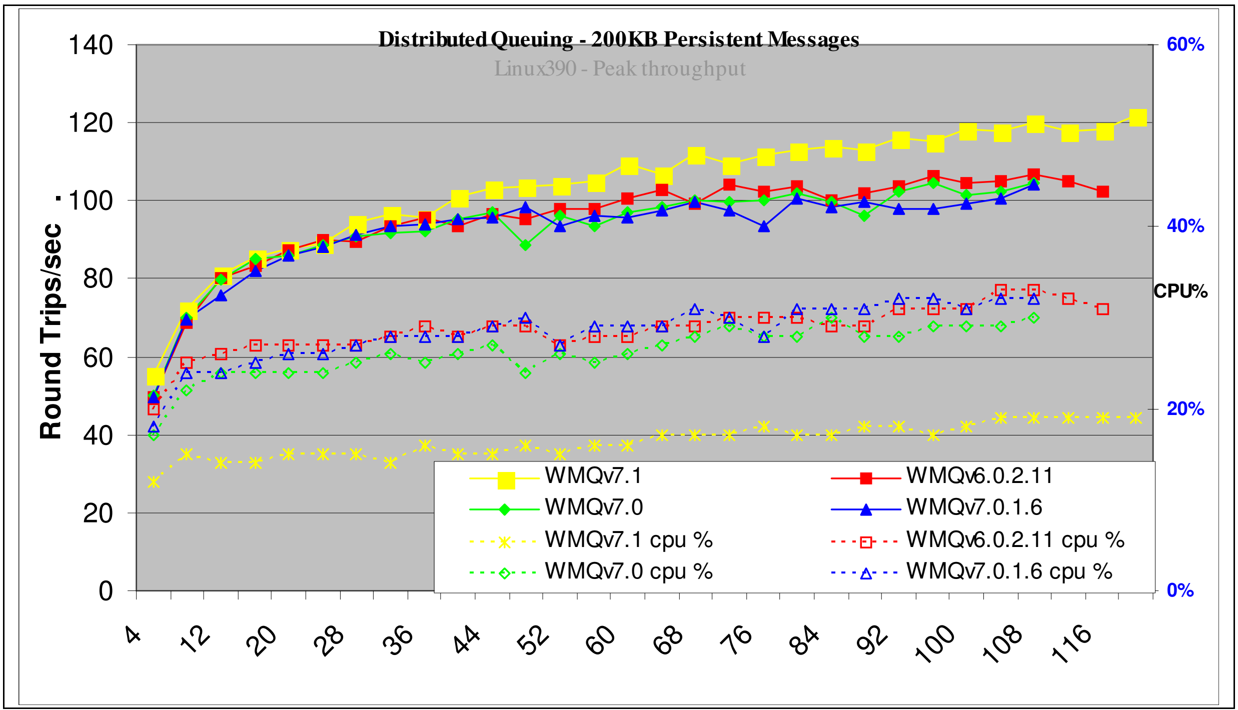


Figure 34 – 200KB persistent messages, distributed queuing

Figure 34 and Table 23 shows that the peak throughput of persistent messages has increased by 17% when comparing version 7.1 to 7.0.1.6 and by 14% when comparing version 7.1 to 6.0.2.11.

Test Name: Distributed Queuing - 200KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	108	107	1.2	33%
WMQv7.0	108	104	1.1	30%
WMQv7.0.1.6	108	104	1.1	32%
WMQv7.1	120	122	1.1	19%

Table 23 – 200KB persistent messages, distributed queuing

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.4 2MB Messages

3.4.1 Local Queue Manager

Figure 35 and Figure 36 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

3.4.1.1 Non-persistent Messages

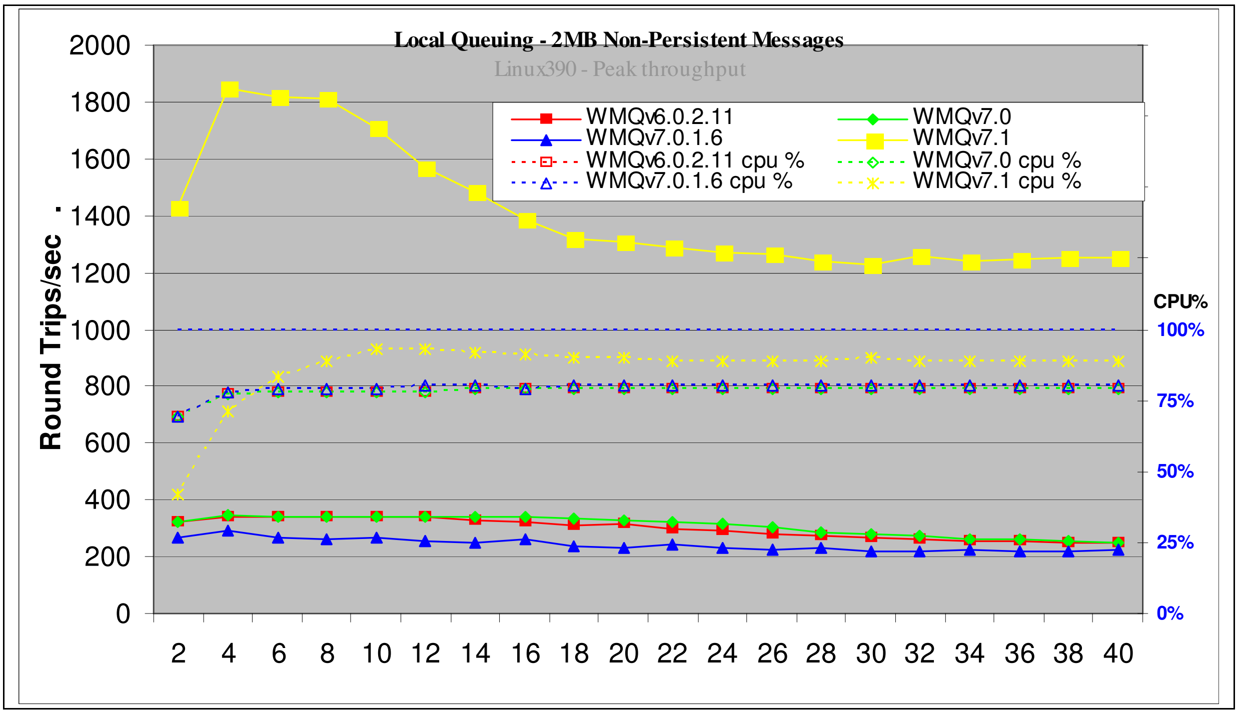


Figure 35 – 2MB non-persistent messages, local queue manager

Figure 35 and Table 24 shows that the peak throughput of non-persistent messages has increased by 500% when comparing version 7.1 to 7.0.1.6 and by 400% when comparing version 7.1 to 6.0.2.11.

Test Name: Local Queuing - 2MB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	4	343	0.012	77%
WMQv7.0	4	344	0.012	77%
WMQv7.0.1.6	4	289	0.014	78%
WMQv7.1	4	1851	0.0042	71%

Table 24 – 2MB non-persistent messages, local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.4.1.2 Persistent Messages

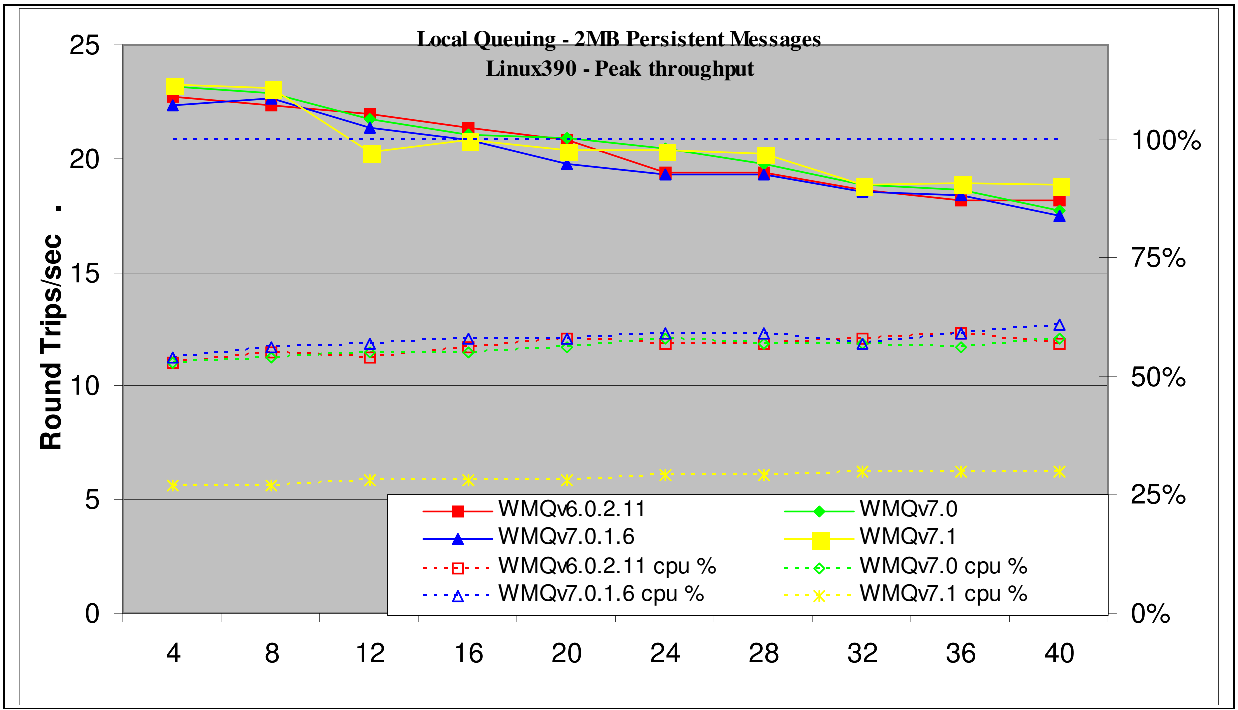


Figure 36 – 2MB persistent messages, local queue manager

Figure 36 and Table 25 shows that the peak throughput of persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11.

Test Name: Local Queuing - 2MB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	4	23	0.17	53%
WMQv7.0	4	23	0.17	53%
WMQv7.0.1.6	8	23	0.38	56%
WMQv7.1	4	23	0.17	27%

Table 25 – 2MB persistent messages, local queue manager

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.4.2 Client Channel

Figure 37 and Figure 38 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

3.4.2.1 Non-persistent Messages

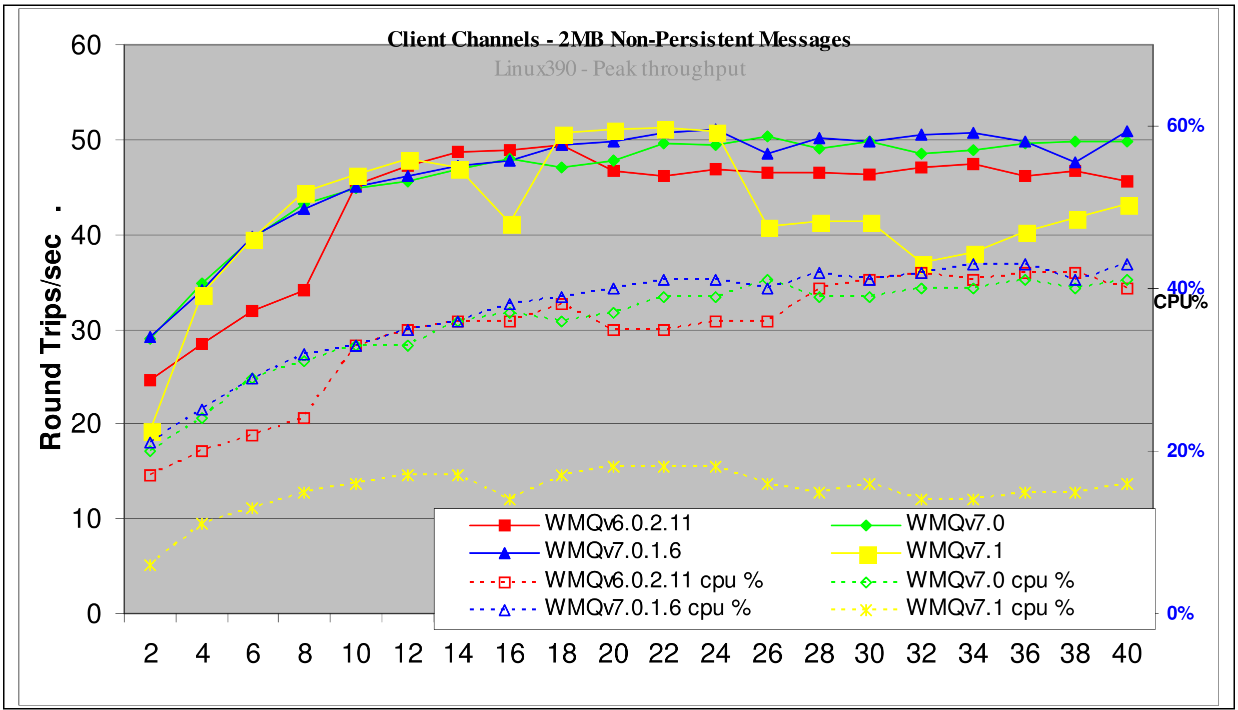


Figure 37 – 2MB non-persistent messages, client channels

Figure 37 and Table 26 shows that the peak throughput of non-persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11.

Test Name: Client Channels - 2MB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	18	49	0.42	38%
WMQv7.0	26	50	0.61	41%
WMQv7.0.1.6	24	51	0.55	41%
WMQv7.1	22	51	0.5	18%

Table 26 – 2MB non-persistent messages, client channels

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.4.2.2 Persistent Messages

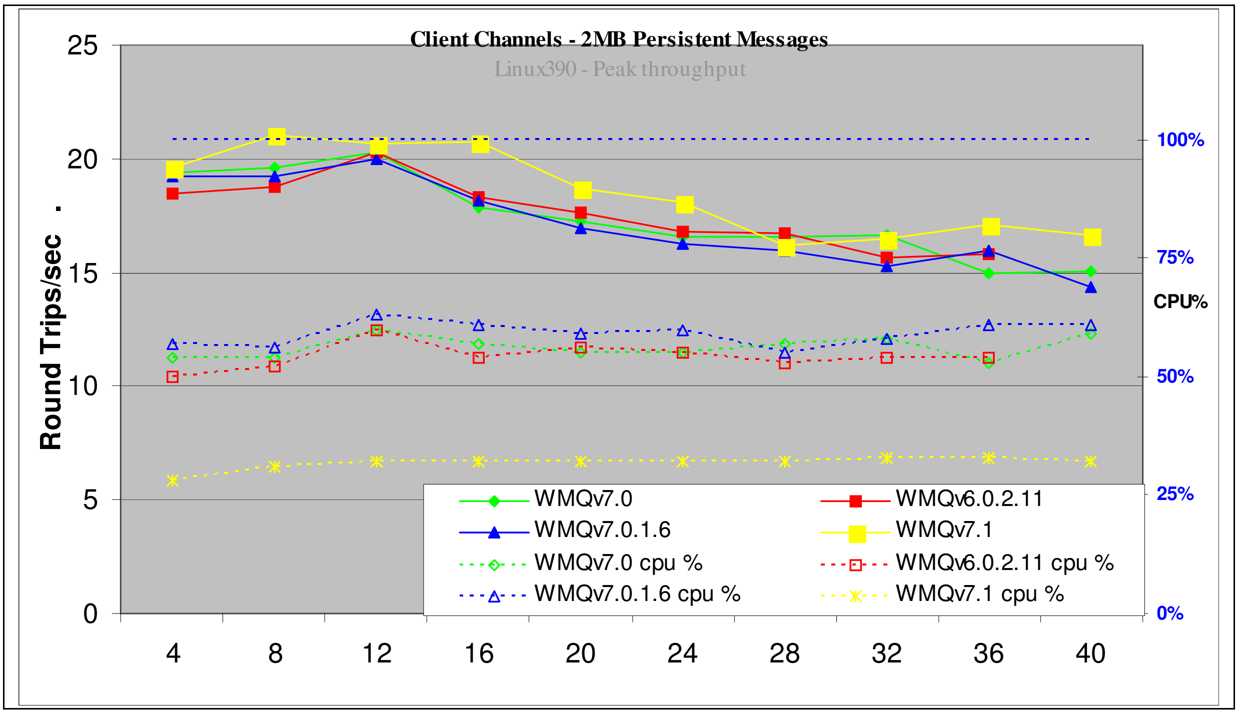


Figure 38 – 2MB persistent messages, client channels

Figure 38 and Table 27 shows that the peak throughput of persistent messages is similar when comparing version 7.1 to 7.0.1.6 and 6.0.2.11.

Test Name: Client Channels - 2MB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	12	20	0.65	60%
WMQv7.0	12	20	0.65	60%
WMQv7.0.1.6	12	20	0.67	63%
WMQv7.1	8	21	0.4	31%

Table 27 – 2MB persistent messages, client channels

Note: *The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time*

3.4.3 Distributed Queuing

Figure 39 and Figure 40 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario.

3.4.3.1 Non-persistent Messages

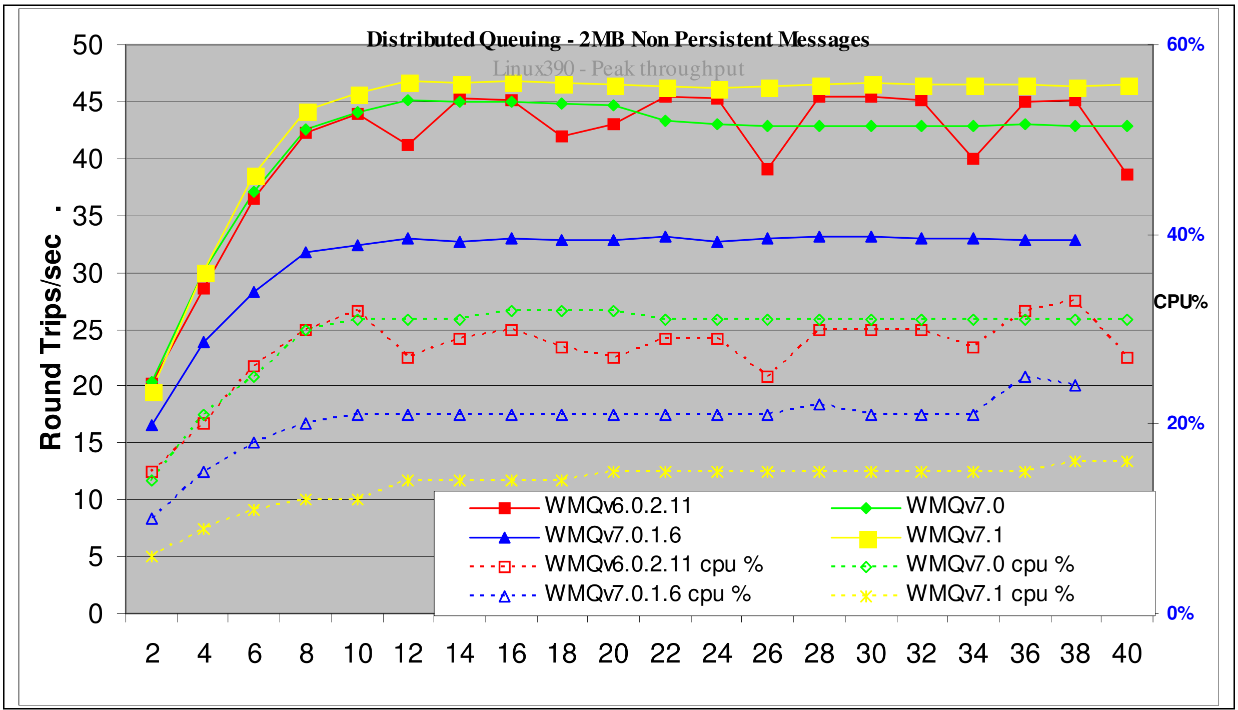


Figure 39 – 2MB non-persistent messages, distributed queuing

Figure 39 and Table 28 shows that the peak throughput of non-persistent messages has increased by 41% when comparing version 7.1 to 7.0.1.6 and is unchanged when comparing version 7.1 to 6.0.2.11.

Test Name: Distributed Queuing - 2MB Non Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	30	46	0.78	30%
WMQv7.0	12	45	0.3	31%
WMQv7.0.1.6	30	33	1	21%
WMQv7.1	12	47	0.29	14%

Table 28 – 2MB non-persistent messages, distributed queuing

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

3.4.3.2 Persistent Messages

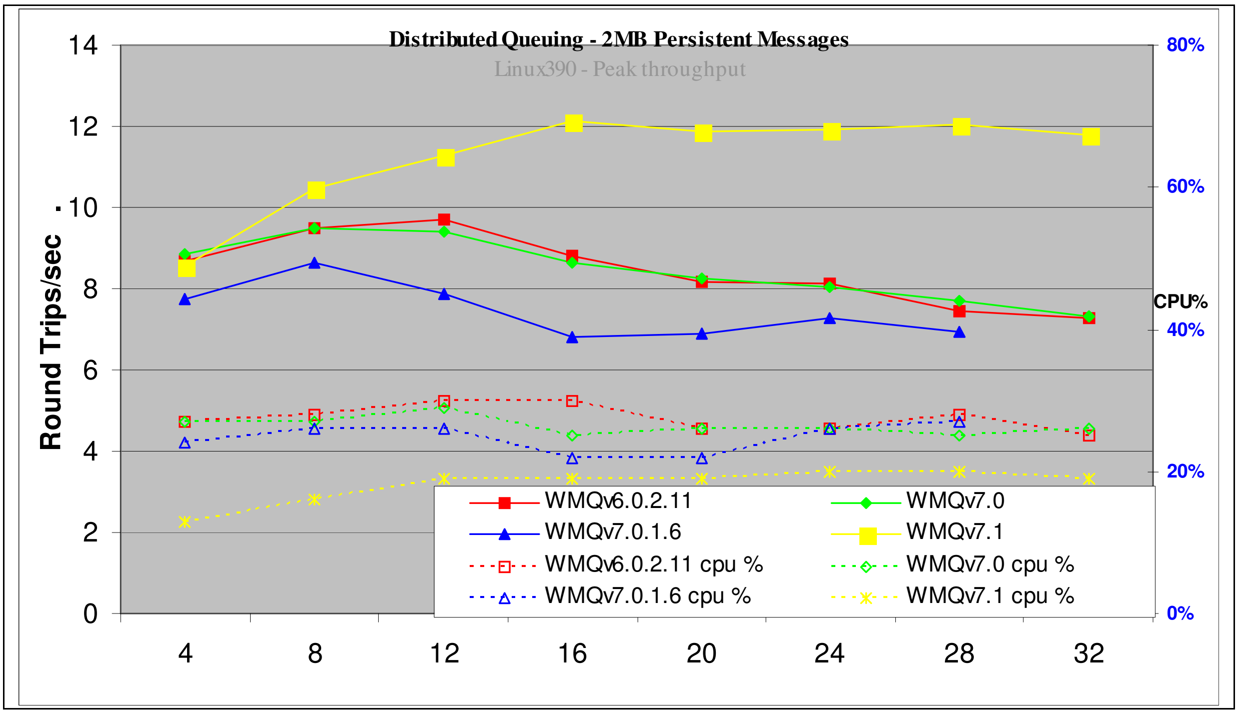


Figure 40 - 2MB persistent messages, distributed queuing

Figure 40 and Table 29 shows that the peak throughput of persistent messages has increased by 40% when comparing version 7.1 to 7.0.1.6 and by 25% when comparing version 7.1 to 6.0.2.11.

Test Name: Distributed Queuing - 2MB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	12	10	1.3	30%
WMQv7.0	8	9	0.89	27%
WMQv7.0.1.6	8	9	1	26%
WMQv7.1	16	12	1.4	19%

Table 29 – 2MB persistent messages, distributed queuing

Note: The numbers in the table above show the peak number of round trips per second, the number of driving applications used, the response time and the server CPU at that time

4 Application Bindings

This report analyzes the message rate 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	Shared
Isolated	Isolated	Isolated
Trusted	Trusted	Trusted
Non Trusted	Shared	Shared

4.1 Local Queue Manager

Figure 41 and **Figure 42** show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

4.1.1 Non-persistent Messages

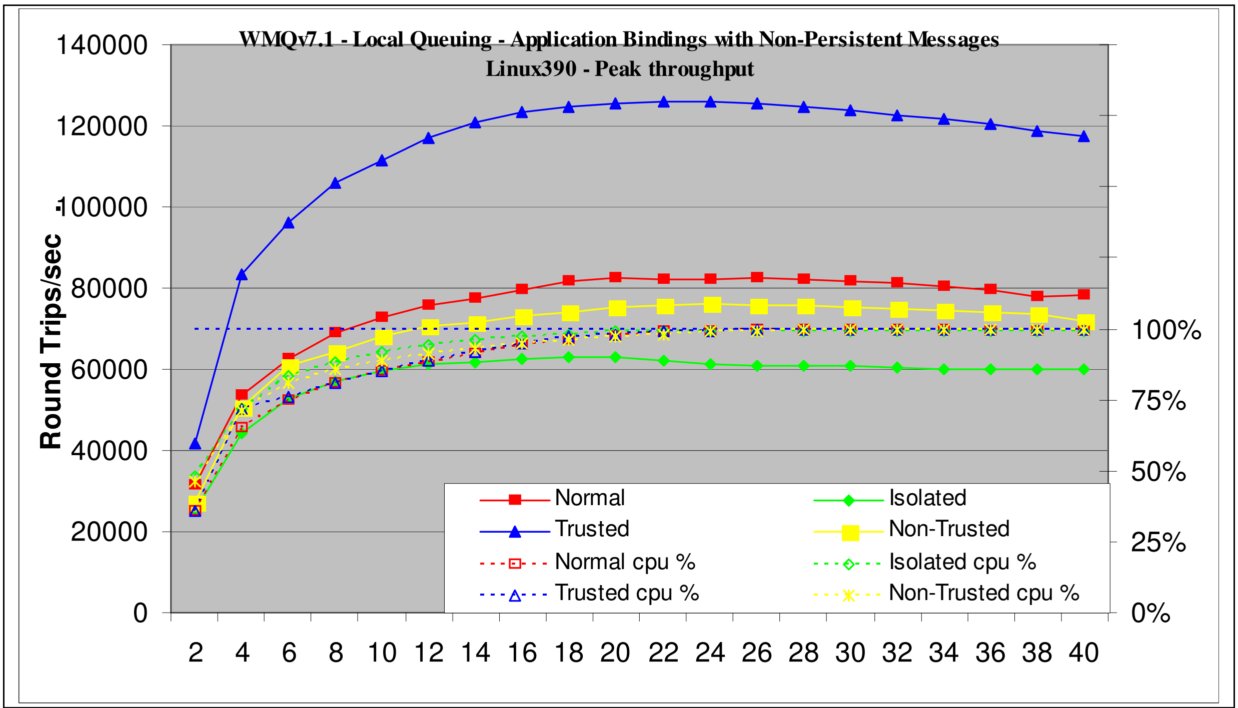


Figure 41 – Application binding, non-persistent messages, local queue manager

Figure 41 and Table 30 show the throughput of non-persistent messages when comparing Normal, Isolated, Trusted and Shared bindings.

Test Name: WMQv7.1 - Local Queuing - Application Bindings with Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
Normal	26	82671	0.00026	100%
Isolated	18	62970	0.00025	98%
Trusted	22	125861	0.00019	99%
Non-Trusted	24	76042	0.00027	99%

Table 30 – Application binding, non-persistent messages, local queue manager

4.1.2 Persistent Messages

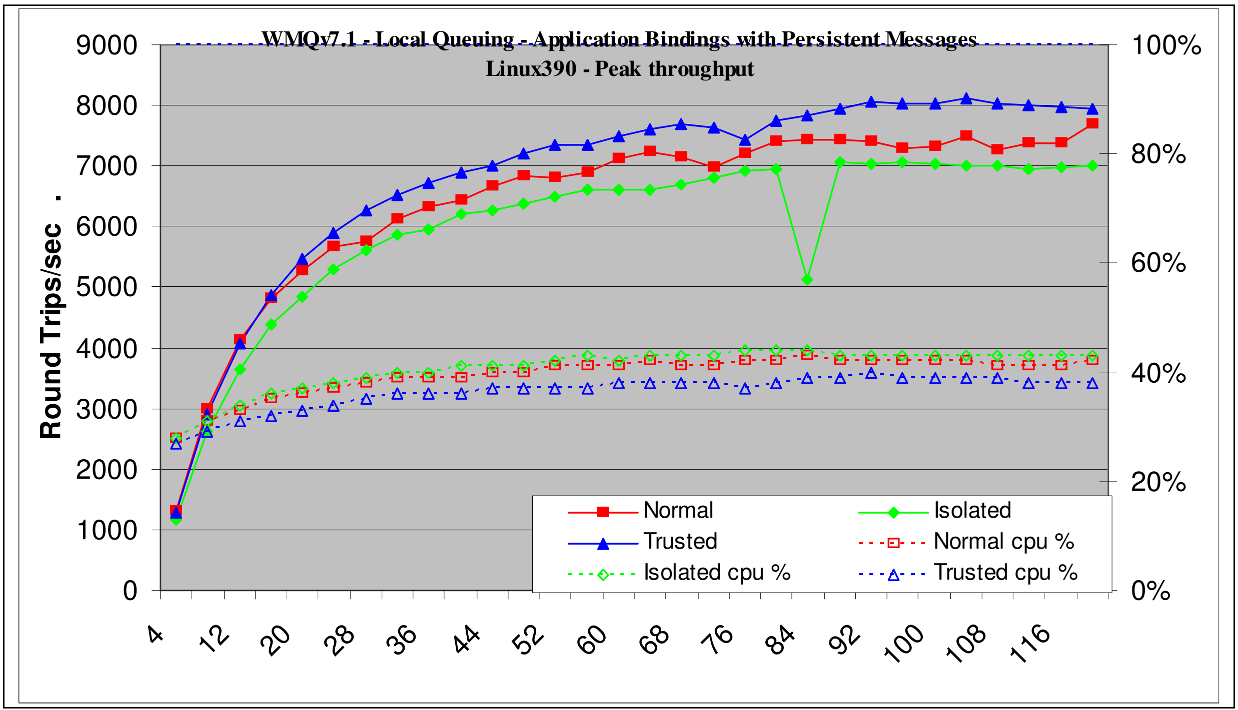


Figure 42 – Application binding, persistent messages, local queue manager

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

Test Name: WMQv7.1 - Local Queuing - Application Bindings with Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
Normal	120	7689	0.018	42%
Isolated	88	7076	0.018	43%
Trusted	104	8118	0.015	39%

Table 31 – Application binding, persistent messages, local queue manager

.

4.2 Client Channels

Figure 43 and **Figure 44** show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

4.2.1 Non-persistent Messages

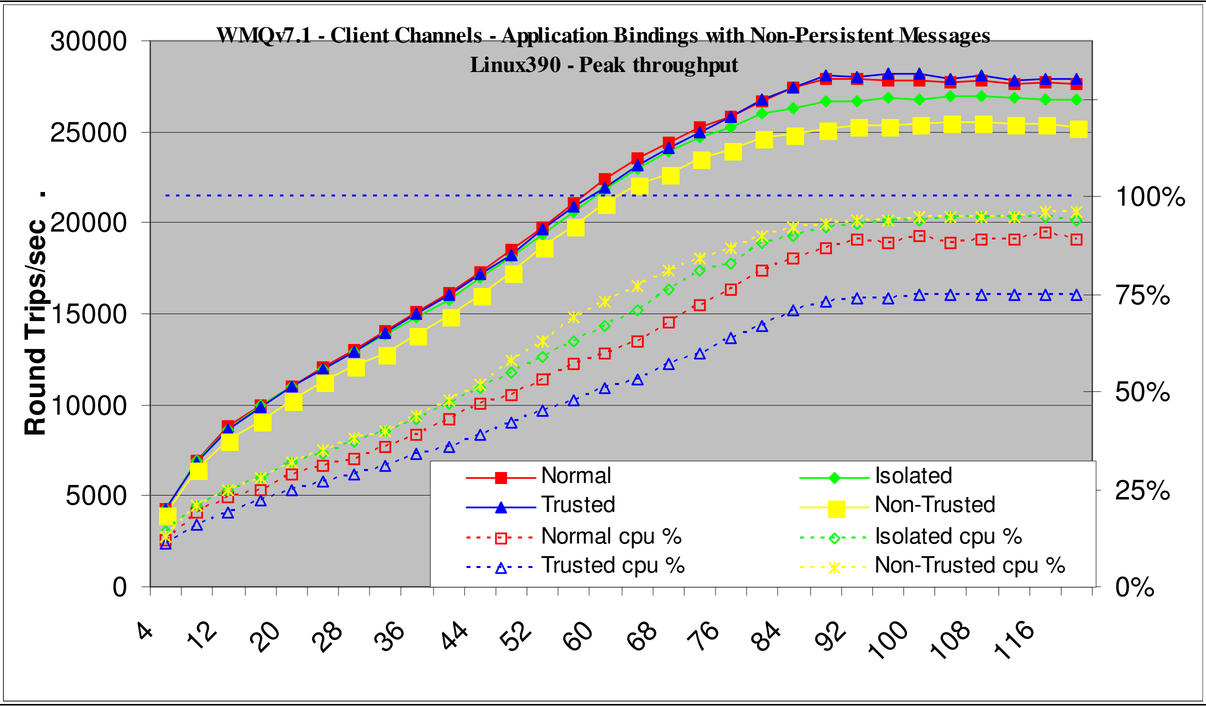


Figure 43 – Application binding, non-persistent messages, client channels

Figure 43 and Table 32 show the peak throughput of non-persistent messages when comparing Normal, Isolated and Trusted bindings.

Test Name: WMQv7.1 - Client Channels - Application Bindings with Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
Normal	88	27907	0.0037	87%
Isolated	104	26936	0.0045	95%
Trusted	96	28192	0.0038	74%
Non-Trusted	104	25512	0.0048	95%

Table 32 – Application binding, non-persistent messages, client channels

4.2.2 Persistent Messages

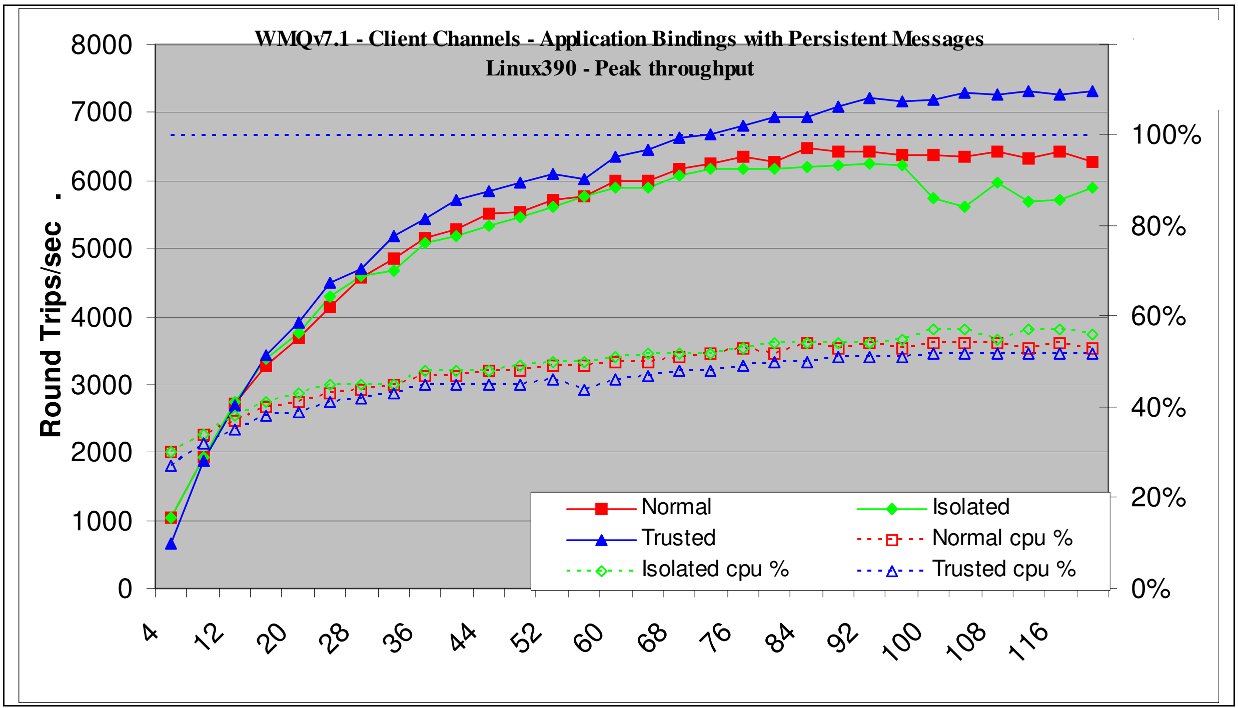


Figure 44 – Application binding, persistent messages, client channels

Figure 44 and Table 33 show the peak throughput of non-persistent messages when comparing Normal, Isolated and Trusted bindings.

Test Name: WMQv7.1 - Client Channels - Application Bindings with Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
Normal	84	6484	0.014	54%
Isolated	92	6256	0.017	54%
Trusted	112	7317	0.018	52%

Table 33 – Application binding, persistent messages, client channels

.

4.3 Distributed Queuing

Figure 44 and **Figure 45** show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario.

4.3.1 Non-persistent Messages

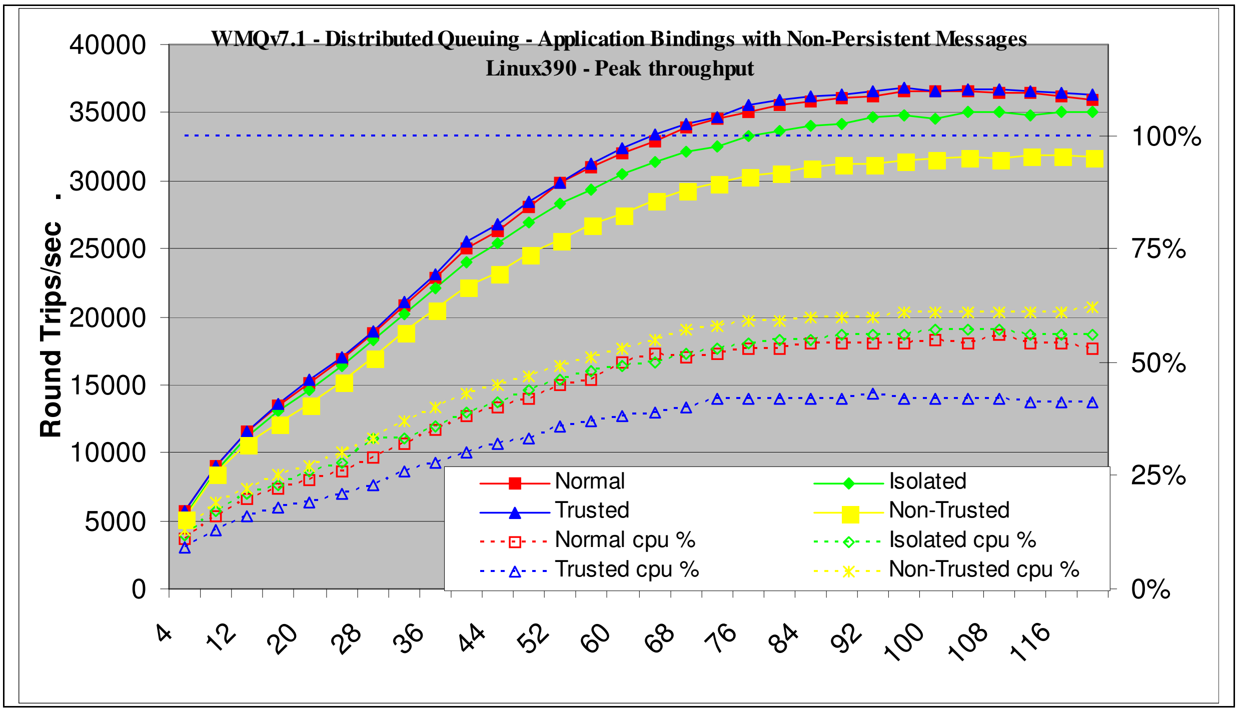


Figure 45 – Application binding, non-persistent messages, distributed queuing

Figure 45 and Table 34 show the peak throughput of non-persistent messages when comparing Normal, Isolated and Trusted bindings.

Test Name: WMQv7.1 - Distributed Queuing - Application Bindings with Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
Normal	104	36545	0.0033	54%
Isolated	104	35096	0.0036	57%
Trusted	96	36807	0.0031	42%
Non-Trusted	116	31883	0.0043	61%

Table 34 – Application binding, non-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.

4.3.2 Persistent Messages

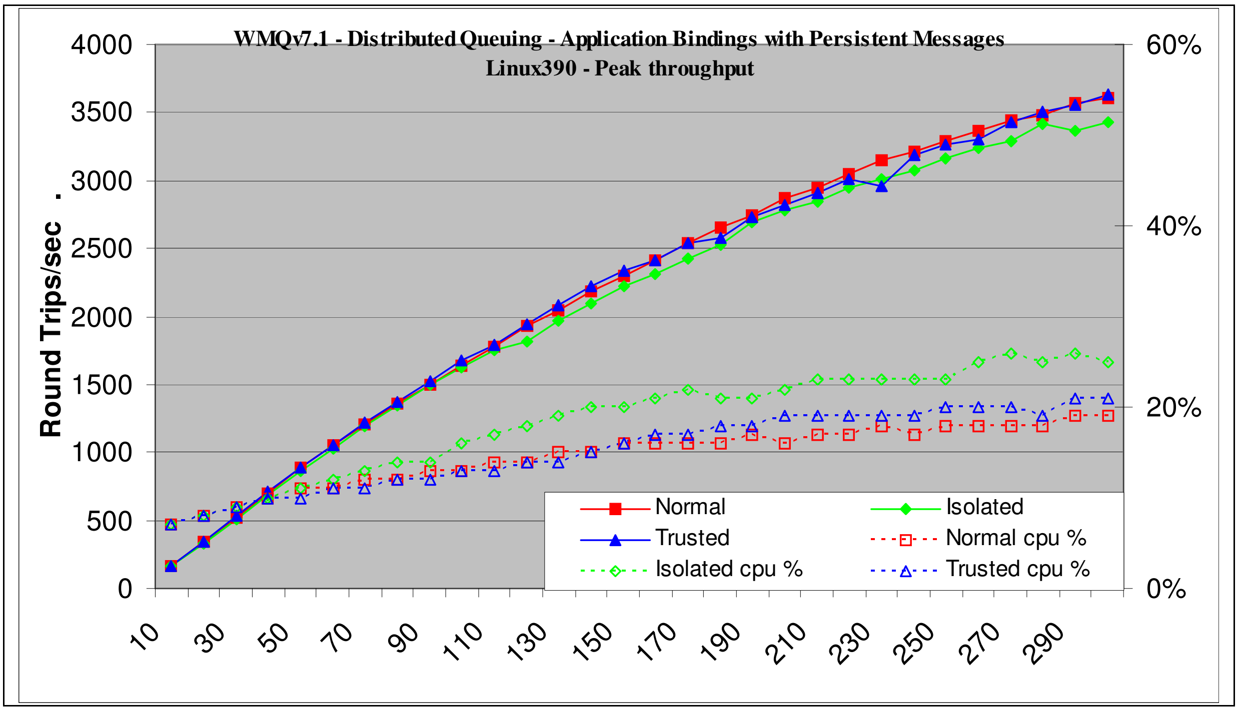


Figure 46 – Application binding, persistent messages, distributed queuing

Figure 46 and Table 35 show the peak throughput of non-persistent messages when comparing Normal, Isolated and Trusted bindings.

Test Name: WMQv7.1 - Distributed Queuing - Application Bindings with Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
Normal	300	3603	0.092	19%
Isolated	300	3424	0.097	25%
Trusted	300	3627	0.1	21%

Table 35 – 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.

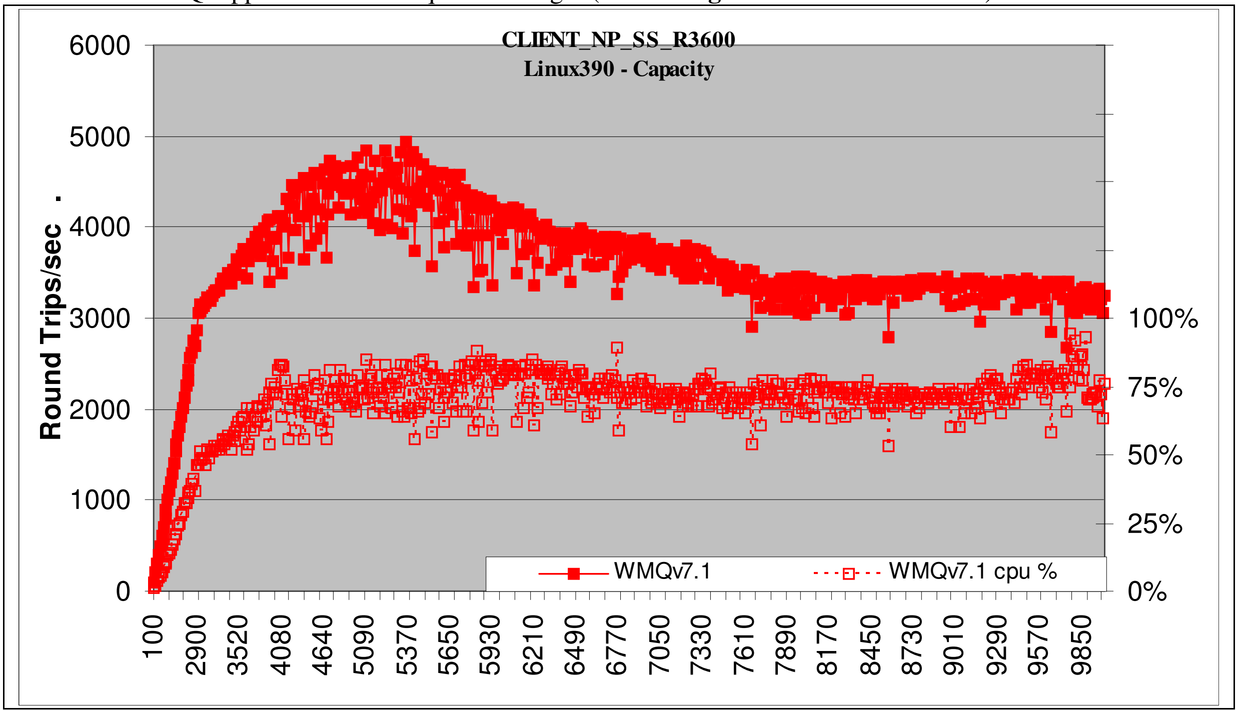
5 Short & Long Sessions

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 47** for an illustration).



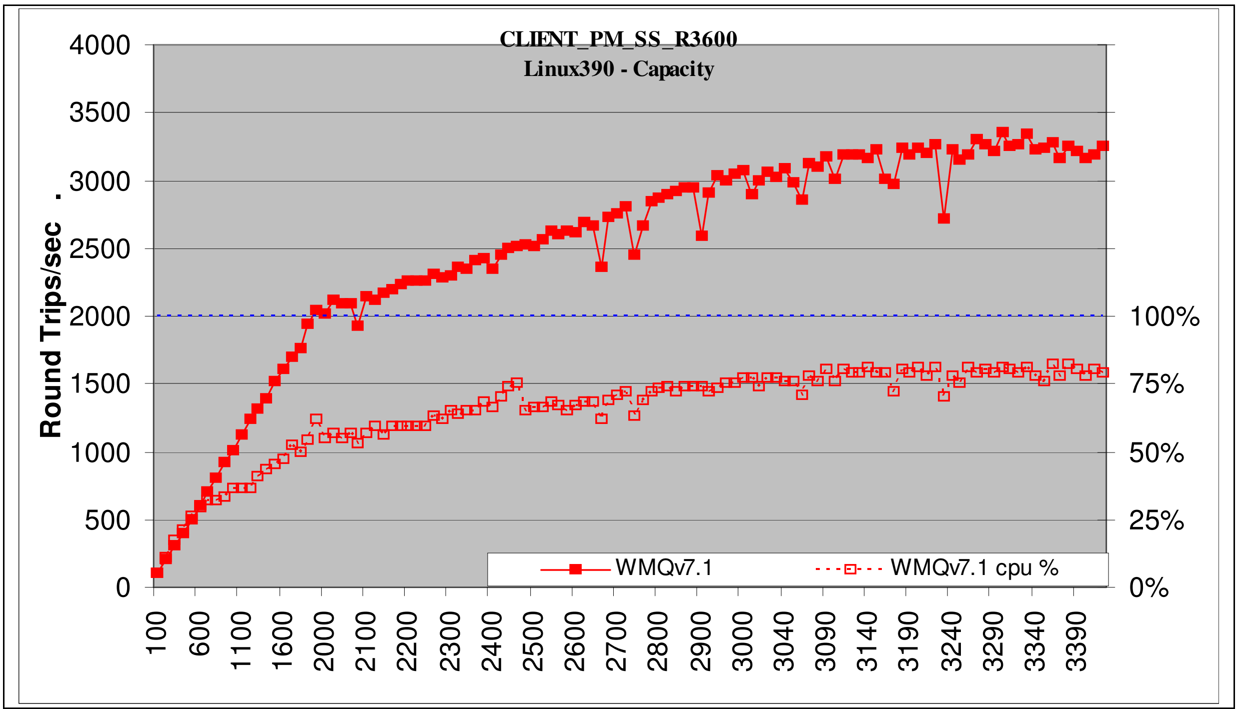


Figure 47 – Short sessions, client channels

Test Name: CLIENT_NP_SS_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv7.1	5360	4943	0.64	83%

Test Name: CLIENT_PM_SS_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv7.1	3300	3348	0.65	81%

Table 36 – 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

No content in this edition.

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 V7.1; 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 a documented external interface and maybe changed or be removed in future releases.

7.1.1 Queue Disk, Log Disk, and Message Persistence

Non-persistent 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 that are 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 non-persistent 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 Non-persistent and Persistent Queue Buffer

The default non-persistent 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 non-persistent 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 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). 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%. 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 non-persistent 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 non-persistent messages.

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 non-persistent 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

Please follow zLinux specific tuning guidelines to apply these values.

/etc/sysctl.conf
net.ipv4.ip_forward = 0
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
kernel.sysrq = 0
kernel.core_uses_pid = 1
net.ipv4.tcp_syncookies = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
net.ipv4.conf.all.accept_redirects = 0
kernel.sem = 500 512000 250 4096
kernel.msgmni = 1024
kernel.shmmni = 4096
kernel.shmall = 2097152
kernel.shmmax = 268435456
fs.file-max = 400000
kernel.pid_max = 120000
net.ipv4.ip_local_port_range = 8192 65535
vm.max_map_count=1966080

7.4 Virtual Memory, Real Memory, & Paging

7.4.1 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.

7.4.2 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

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.

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.

Non-persistent 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 various threads for processing non-persistent messages and persistent messages. Each server thread performed the sequence of actions as outlined in the test scenario illustrations.

Non-persistent 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.

8.1.2.3 Analysis techniques

In the overview section, the percentage throughput comparison used the area under the graph as an alternative method of interpreting the performance data. Elsewhere, the percentage throughput comparison used the peak throughputs found in the tables associated with the graphs. The area under the curve is favoured in this instance as it gives a much more general performance indicator.

NB: Locking improvements in WMQv7.1 have improved the right hand side of the graphs but came with path length costs that may affect the rate of growth on left hand side of the graph when there is only a small number of parallel applications.

8.2 Hardware specification

IBM zSeries 990:	Server system (Device under test). LPAR with VM hypervisor
Model:	2084-331
Architecture:	2 way SMP (VM has 31 processors with 4 dedicated to this VM/Linux LPAR)
Memory (RAM):	2GB of virtual memory. (VM system has 6GB of main storage + 1GB of expanded)
Disk:	3390-9 DASD on Shark 2105-800
Network:	1Gbit Ethernet Adapter

IBM x3850:	Driver system
Model:	x3850 M2 8864 4RG
Processor:	2.93GHz Intel Xeon x7350
Architecture:	2 x quad core CPU
Memory (RAM):	32GB
Disk:	2 Internal 16bit SCSI (9GB each, 1 O/S, swap)
Network:	1Gbit Ethernet Adapter

8.3 Software

Linux on zSeries:	SUSE Linux Enterprise Server 10 (s390x)
MQSeries:	Version 6.0.2.11, Version 7.0, Version 7.0.1.6, Version 7.1

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** => non-persistent 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.
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).
FREE	Free memory as reported by IOSTAT