WebSphere MQ Linux v7.1
Performance Evaluations

Version 1.2

February 2012

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 Linux 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

wanting to understand the performance characteristics of *WebSphere MQ for Linux v7.1*. 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 v7.1.

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

The SupportPac was designed for people who:

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

A xSeries 3650M3 box containing two 6 core 2.80GHz Intel Xeon CPUs and 32GB 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.

How this document is arranged

Pages: 1-15

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.

Large Messages

Pages: 21-42

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".

Application Bindings

Page: 43-48

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

Performance and Capacity Limits

Pages: 51 - 53

Tuning Recommendations

Pages: 53- 58

Tuning guidance specific to v7.1 on Linux

Measurement Environment

Pages: 58 59

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: 60

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

CONTENTS

TABLES

FIGURES

1 Overview

WebSphere MQ v7.1 on Linux 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 93% better than v6.0.2.11
- For 2K persistent messages v7.1 is 48% better than v6.0.2.11

- For 2K non-persistent messages v7.1 is 148% better than v7.0
- For 2K persistent messages v7.1 is 82% better than v7.0

- For 2K non-persistent messages v7.1 is 124% better than v7.0.1.6
- For 2K persistent messages v7.1 is 88% better than v7.0.1.6

IBM WebSphere MQ V7.1 can utilise more cpu cores than previous releases. With the Requester/Responder model using a single queue, higher throughput could be achieved on an 8 core machine than a 12 core machine. Measuresments are documented in 2.1.4 and 3.2.1.2 that show 12 cores provide higher throughput than 8 cores.

IBM WebSphere MQ V7.1 adds multicast as a new alternative for publish/subscribe configurations, ideal for rapidly distributing messages to large numbers of subscribers. Compared to publish/subscribe in the previous release, WebSphere MQ V7.1 offers performance improvements of 500% or more for distributing 256 byte messages to multiple subscribers. These results are discussed in chapter 10.

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

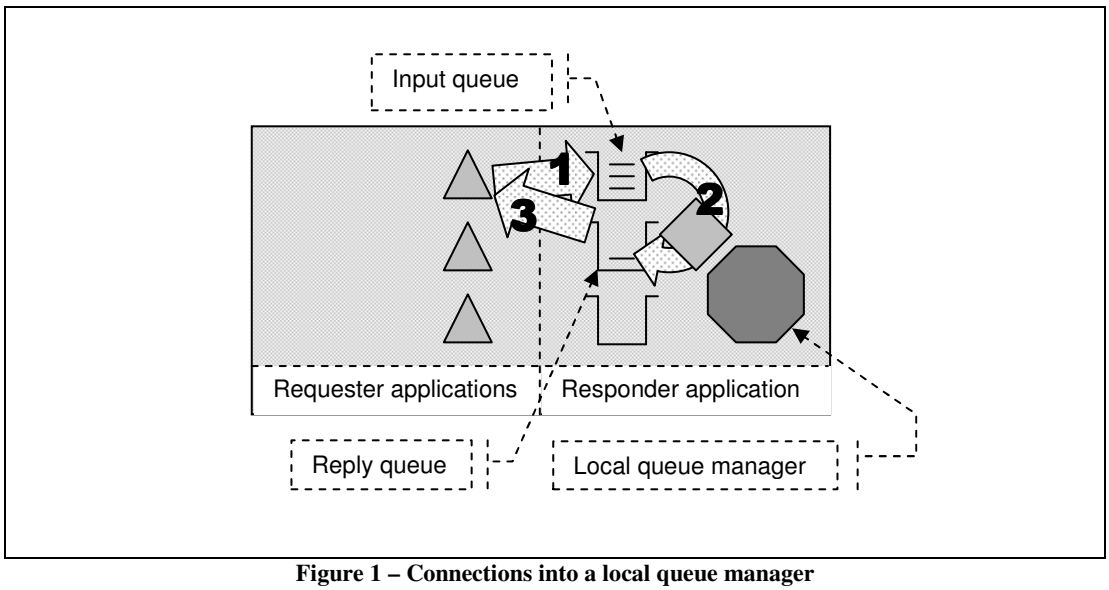


Figure 1 – Connections into a local queue manager

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).

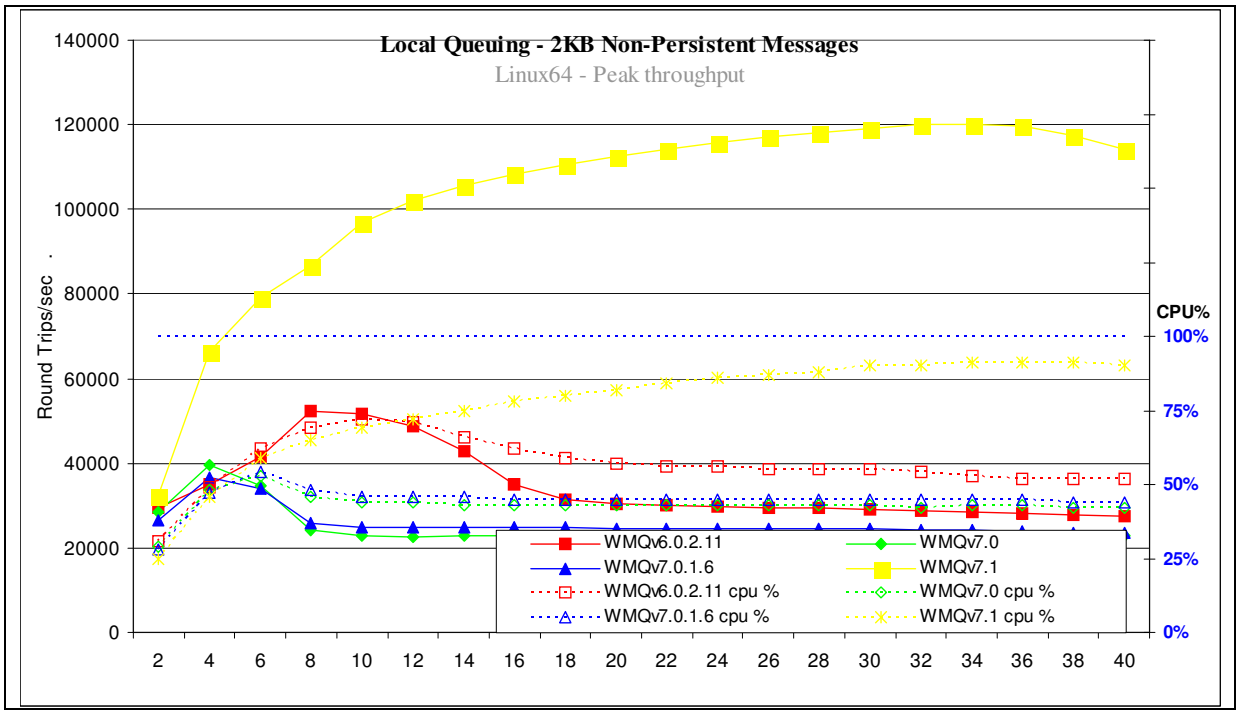


Figure 2 – Performance headline, non-persistent messages and local queue manager.

Figure 2 and Table 1 show that the throughput of non-persistent messages has increased by 228% when comparing version 7.1 to 7.0.1.6 and 129% 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	52392	0.00018	69%
WMQv7.0	4	39627	0.00012	48%
WMQv7.0.1.6	4	36630	0.00013	47%
WMQv7.1	34	120090	0.0003	91%

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

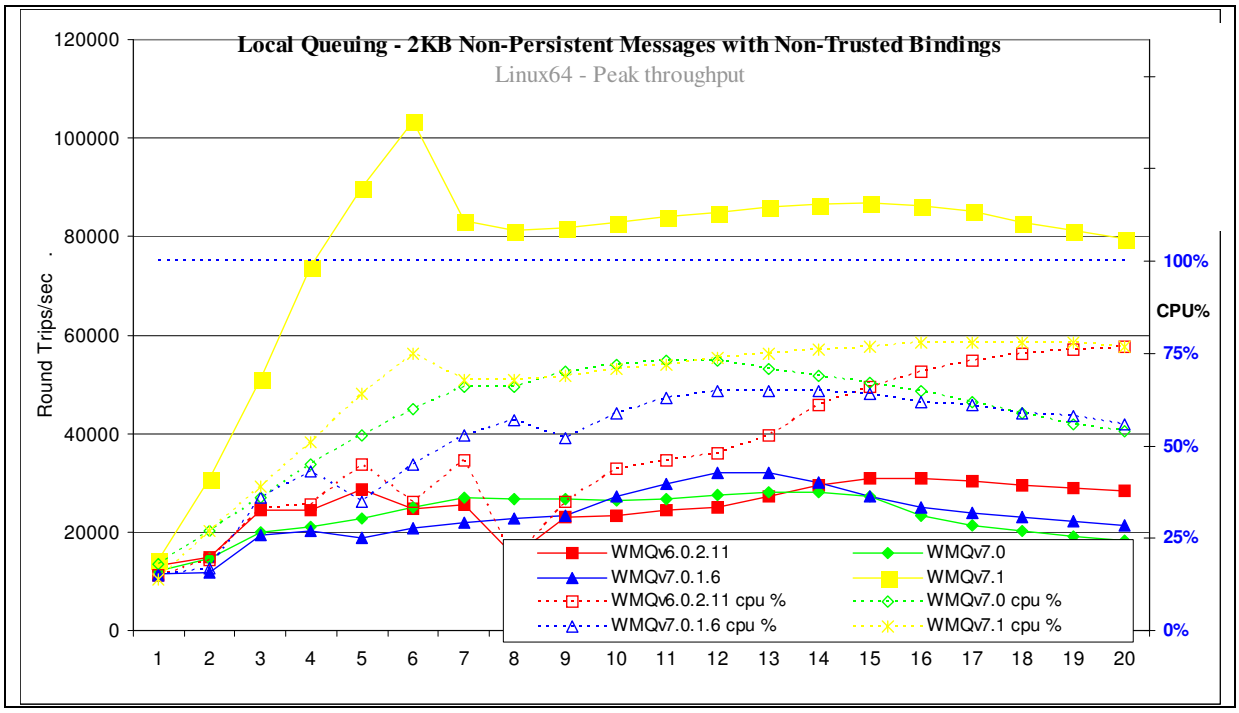


Figure 3 - Performance headline, non-persistent, non-trusted messages and local queue manager.

Figure 3 and Table 2 shows that the throughput of non-persistent, non-trusted messages (shared bindings - MQIBINDTYPE=NORMAL) has increased by 223% when comparing version 7.1 to 7.0.1.6 and 233% 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	15	30985	0.00021	66%
WMQv7.0	14	28230	0.00053	69%
WMQv7.0.1.6	13	31957	0.00029	65%
WMQv7.1	6	103322	0.00008	75%

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

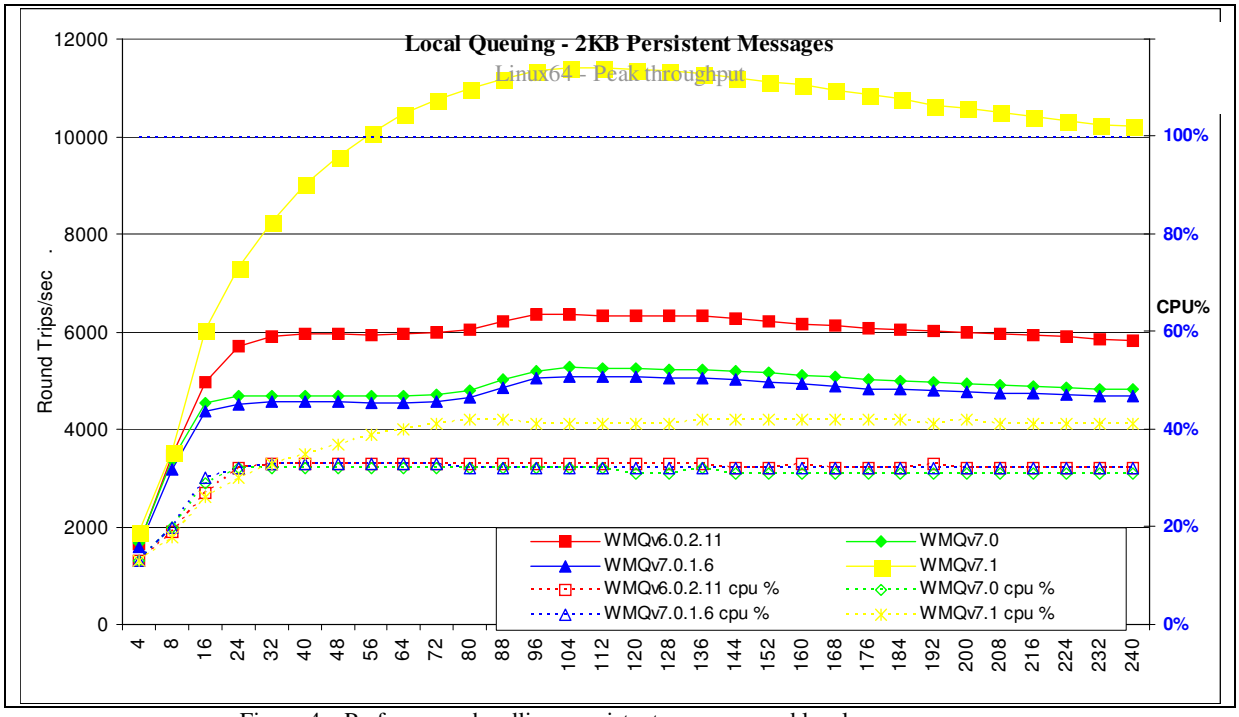


Figure 4 – Performance headline, persistent messages and local queue manager

Figure 4 and Table 3 shows that the throughput of persistent messages has increased by 124% when comparing version 7.1 to 7.0.1.6 and 80% 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	104	6361	0.019	33%
WMQv7.0	104	5263	0.023	32%
WMQv7.0.1.6	112	5089	0.025	32%
WMQv7.1	104	11418	0.01	41%

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

2.1.4 Scalability – Local non Persistent

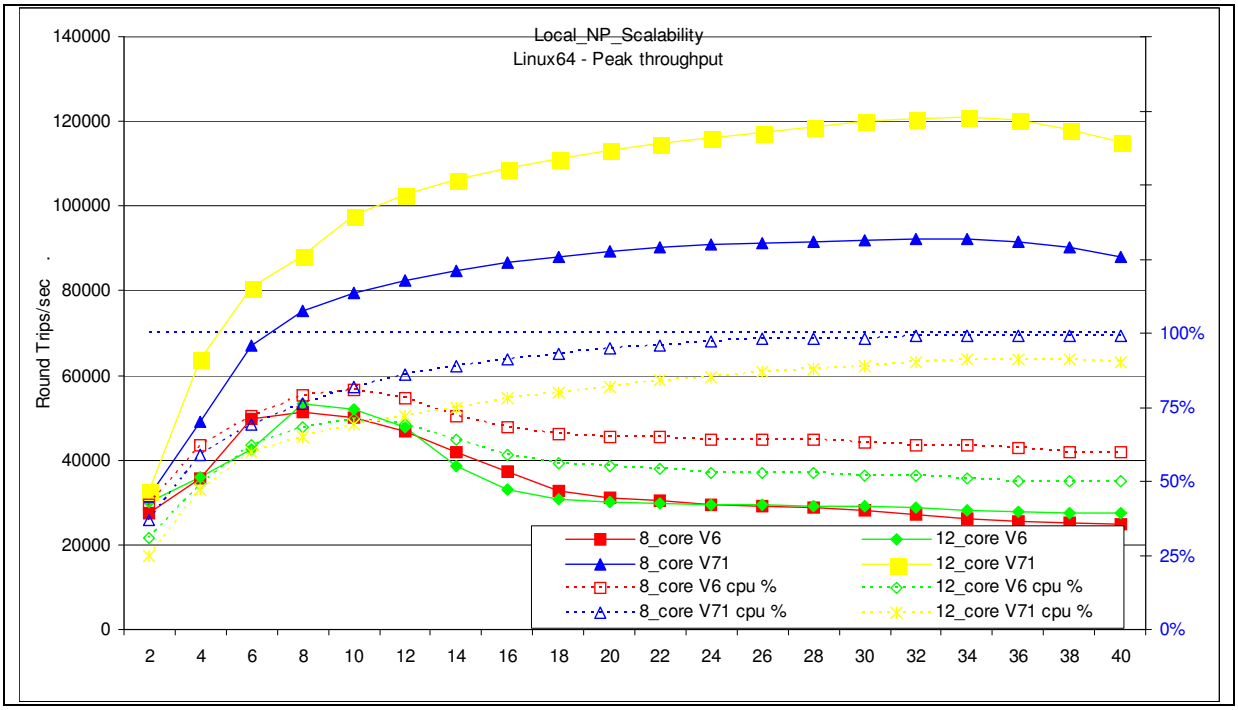


Figure 5 – Scalability, Local Queue manager, non-persistent messages

Test Name: Local_NP_Scalability	Apps	Round Trips/Sec	Response time (s)	CPU
8_core V6	8	51276	0.00017	79%
12_core V6	8	53408	0.00017	68%
8_core V71	32	92208	0.00041	99%
12_core V71	34	121098	0.0003	91%

Table 4 – Scalability, Local Queue manager, non-persistent messages

The throughput comparison of 8 core and 12 cores for MQ V 6.0.2.10 messages is not significantly different. The throughput comparison for MQ V7.1 with 12 cores is 27% larger with the peak difference being 31% larger. The significant improvement over previous levels is maximised in scenarios where messages all go through a single queue

2.2 Client Channels Test Scenario

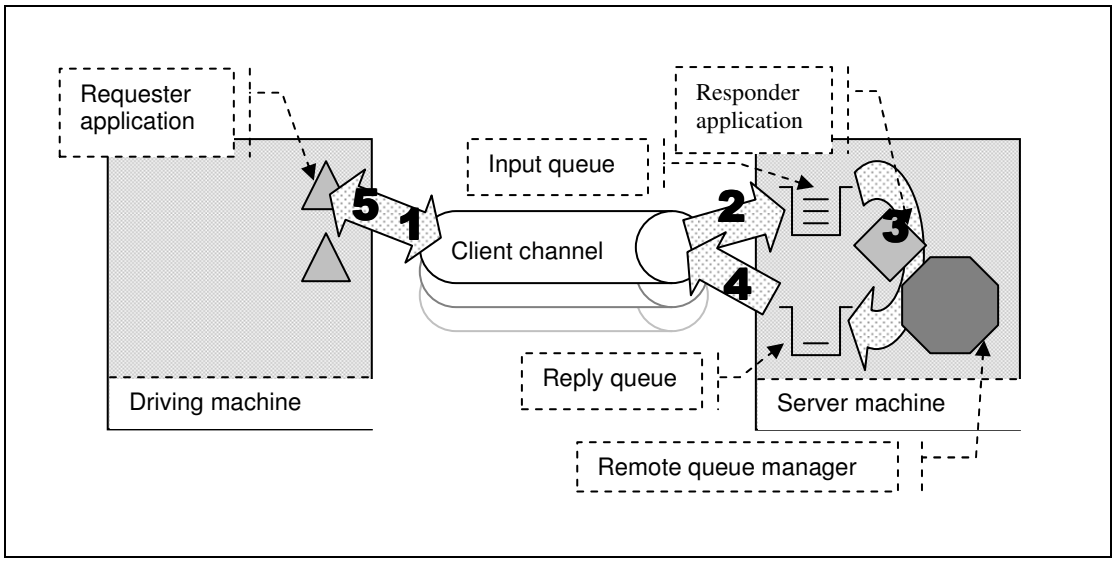


Figure 6 – MQI-client channels into a remote queue manager

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 7, **Figure 8** and Figure 9 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 6 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).

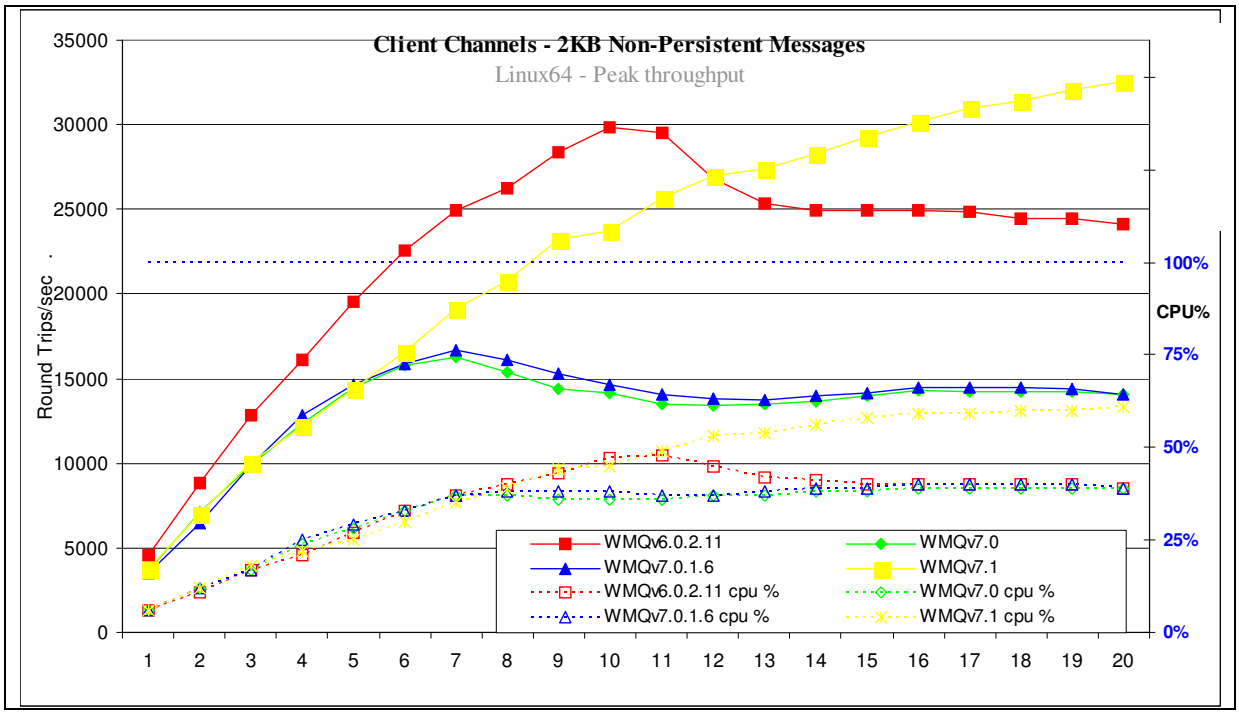


Figure 7 – Performance headline, non-persistent messages and client channels

Figure 7 and Table 5 show that the throughput of non-persistent messages has increased by 96% when comparing version 7.1 to 7.0.1.6.

Test Name: Client Channels - 2KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	10	29882	0.00038	47%
WMQv7.0	7	16293	0.0005	37%
WMQv7.0.1.6	7	16644	0.0005	37%
WMQv7.1	20	32569	0.00072	61%

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

2.2.2 Non-persistent Messages – Non-Trusted Client Channels

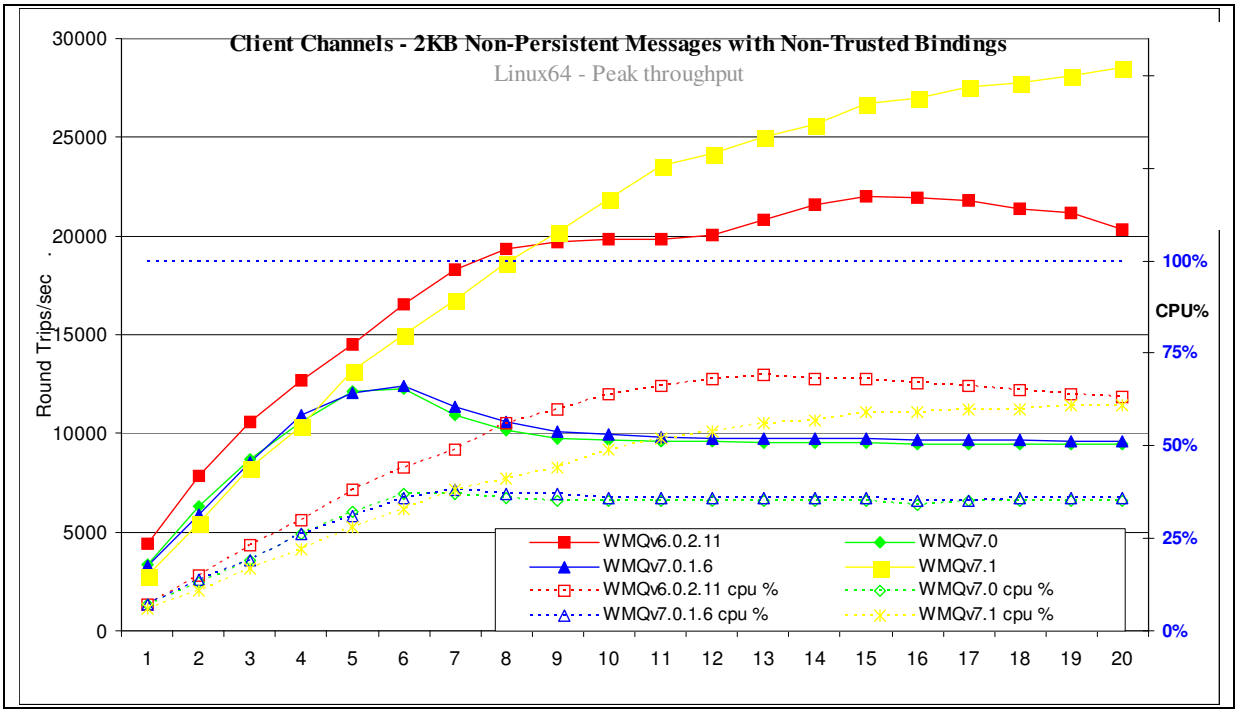


Figure 8 – Performance headline, non-persistent messages with non-trusted client channels

Figure 8 and Table 6 shows that the throughput of non-persistent, non-trusted messages (shared bindings - MQIBINDTYPE=NORMAL) has increased by 130% when comparing version 7.1 to 7.0.1.6 and by 30% 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	15	22000	0.00078	68%
WMQv7.0	6	12270	0.00058	37%
WMQv7.0.1.6	6	12417	0.00056	36%
WMQv7.1	20	28516	0.00085	61%

Table 6 – 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

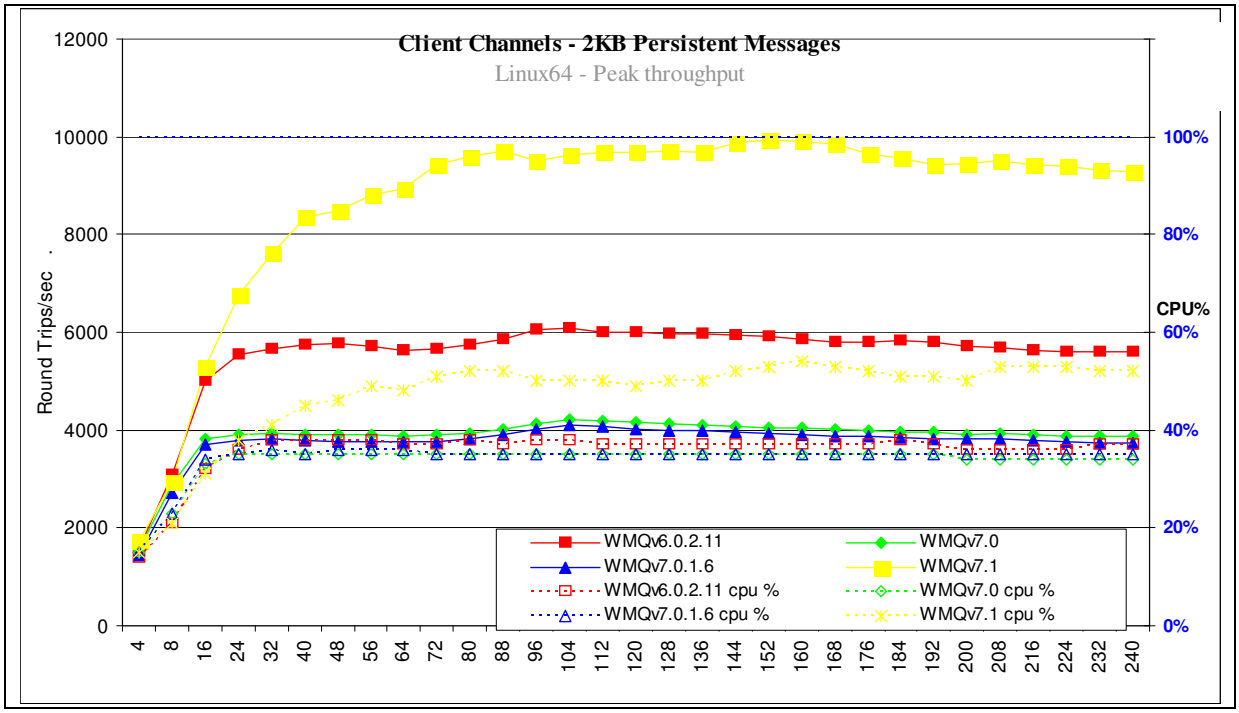


Figure 9 – Performance headline, persistent messages and client channels

Figure 9 and Table 7 shows that the throughput of persistent messages has increased by 143% when comparing version 7.1 to 7.0.1.6 and by 63% 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	104	6089	0.02	38%
WMQv7.0	104	4219	0.029	35%
WMQv7.0.1.6	104	4092	0.03	35%
WMQv7.1	152	9943	0.018	53%

Table 7 – 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.

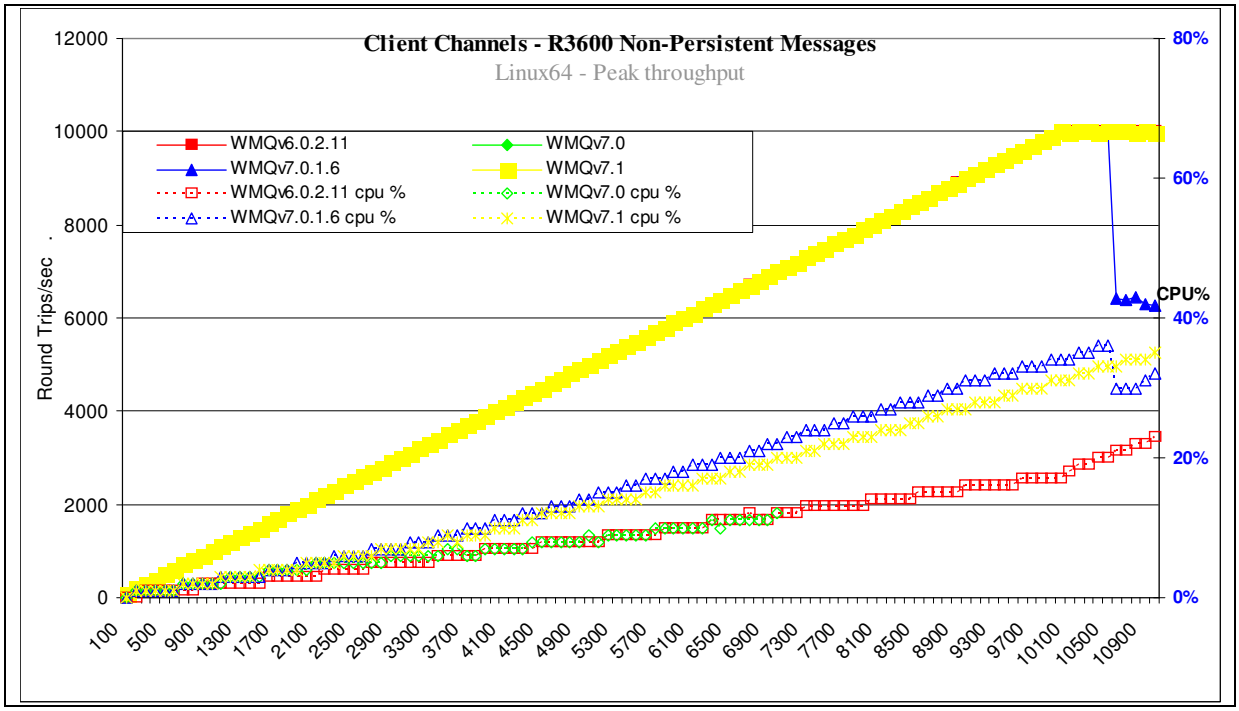


Figure 10 – 1 round trip per driving application per second, client channels and non-persistent messages

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

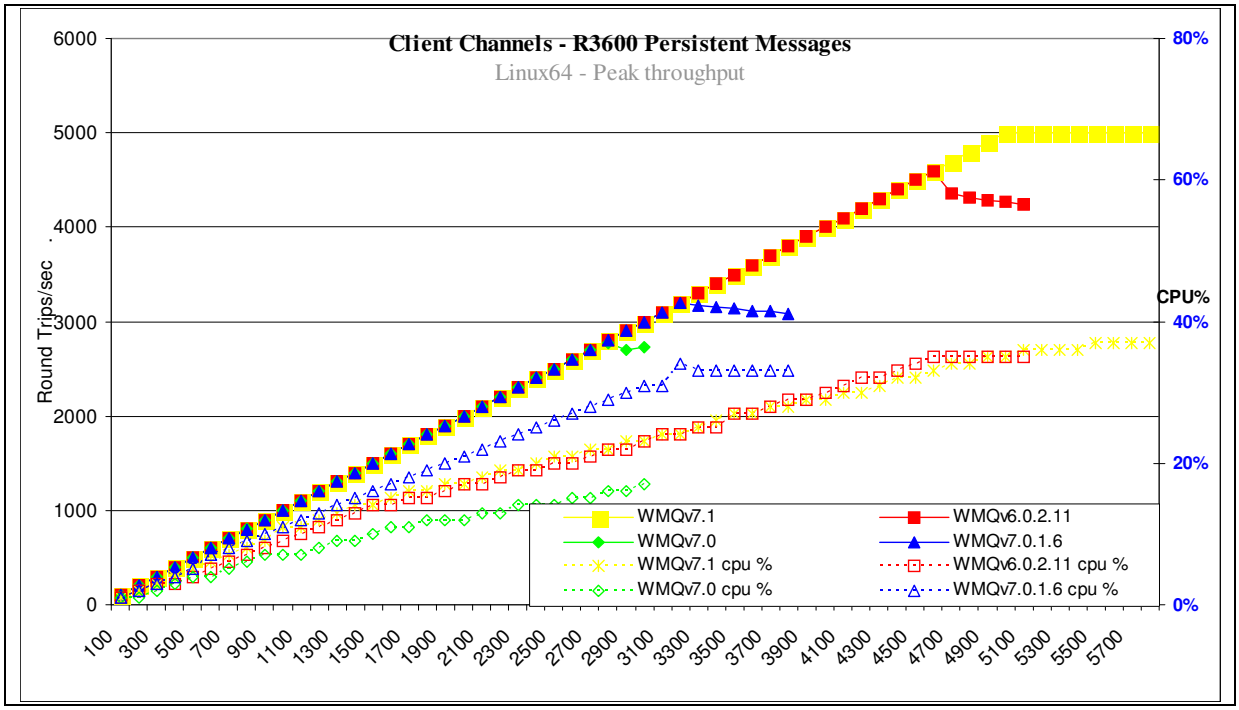


Figure 11 – 1 round trip per driving application per second, client channels, persistent messages

Figure 10, Figure 11 and Table 8 show that the throughput of non-persistent messages has not changed when comparing version 7.1 to 7.0.1.6 and to 6.0.2.11. It also shows that the throughput of persistent messages has increased by 56% when comparing version 7.1 to 7.0.1.6 and by 9% when comparing version 7.1 to 6.0.2.11

Test Name: Client Channels - R3600 Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	10000	9998	0.00056	17%
WMQv7.0	7000	6998	0.00096	12%
WMQv7.0.1.6	10400	9997	0.0059	36%
WMQv7.1	10300	9996	0.00098	32%

Test Name: Client Channels - R3600 Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	4600	4598	0.0082	35%
WMQv7.0	2700	2699	0.098	15%
WMQv7.0.1.6	3200	3198	0.95	34%
WMQv7.1	5000	4998	0.0053	35%

Table 8 – 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

2.2.5 SSL

The following diagram shows how throughput varies depending on the cipher selected. The top line (using the standard 2KB message CLNP test) in not encrypted. The other lines show a selection of the available ciphers.

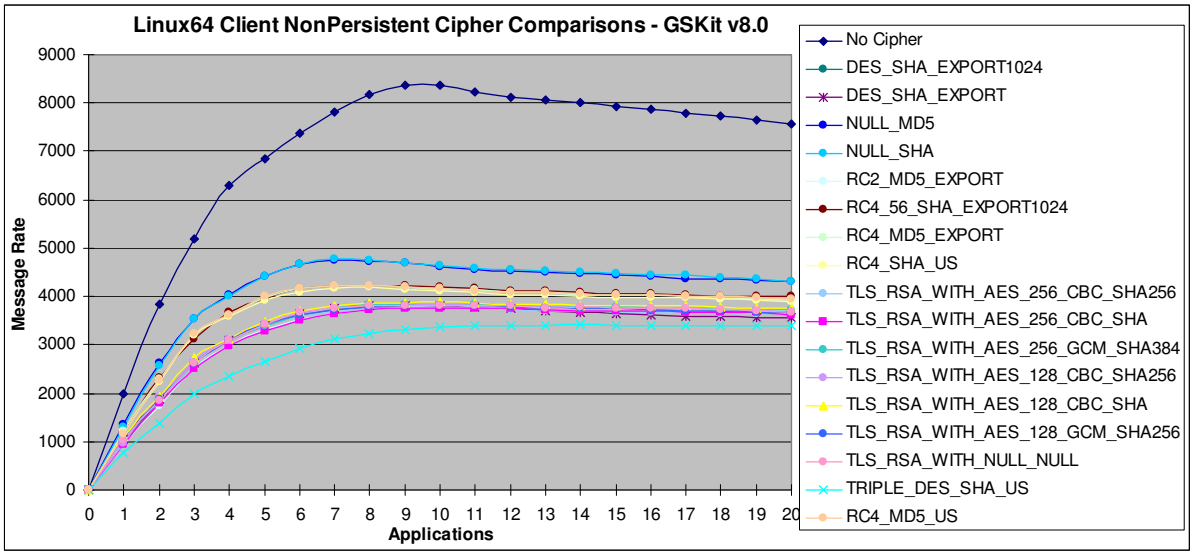


Figure 12 – Client non-persistent message rates with various SSL ciphers

The area under the curve values for each cipher are used to compile an ordered table of these ciphers and can be used as a guide to expected performance degradations that can be expected when these ciphers are used. The unencrypted value is used to rate the other values as a percentage. Thus for the SSL cipher *NULL_SHA*, the expected message rate would be approximately 59% of the unencrypted rate.

Linux64 Cipher Comparisons	CLNP
No Cipher	100%
NULL_SHA	59%
NULL_MD5	58%
RC4_56_SHA_EXPORT1024	53%
RC4_MD5_US	53%
RC4_MD5_EXPORT	53%

Linux64 Cipher Comparisons	CLNP
RC4_SHA_US	52%
TLS_RSA_WITH_AES_128_CBC_SHA	49%
TLS_RSA_WITH_AES_256_GCM_SHA384	48%
TLS_RSA_WITH_NULL_NULL	48%
TLS_RSA_WITH_AES_128_CBC_SHA256	48%
TLS_RSA_WITH_AES_128_GCM_SHA256	48%
DES_SHA_EXPORT1024	47%
RC2_MD5_EXPORT	47%
TLS_RSA_WITH_AES_256_CBC_SHA256	47%
DES_SHA_EXPORT	47%
TLS_RSA_WITH_AES_256_CBC_SHA	47%
TRIPLE_DES_SHA_US	41%

Table 9 – Ordered relative SSL Client cipher performance

2.3 Distributed Queuing Test Scenario

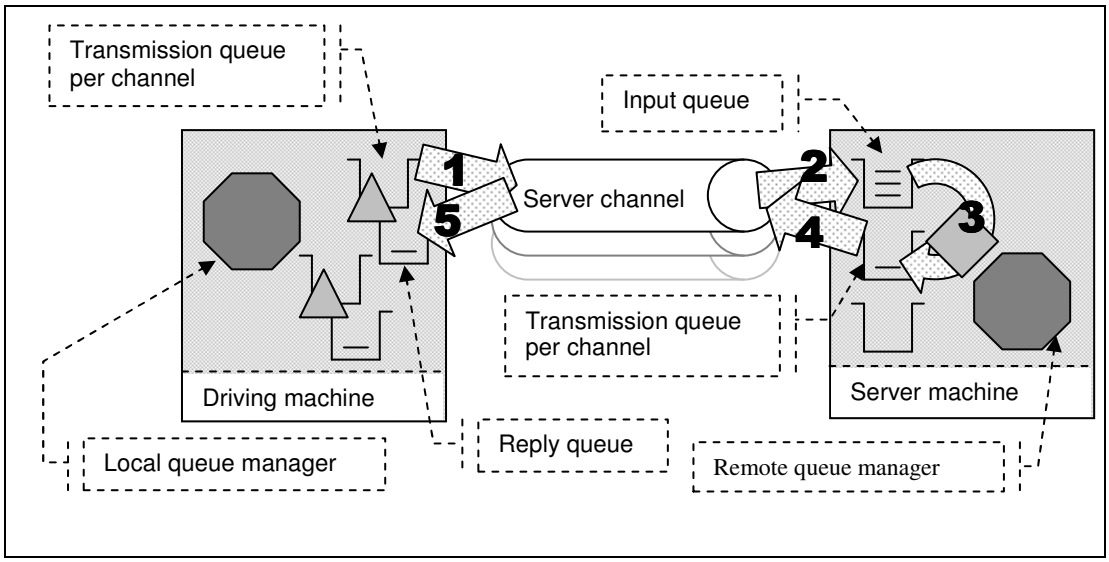


Figure 13 – Server channels between two queue managers

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 14, **Figure 14** and Figure 16 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 13 on the previous page) and WebSphere MQ (versions 7.1, 7.0.1.6, 7.0 and 6.0.2.11).

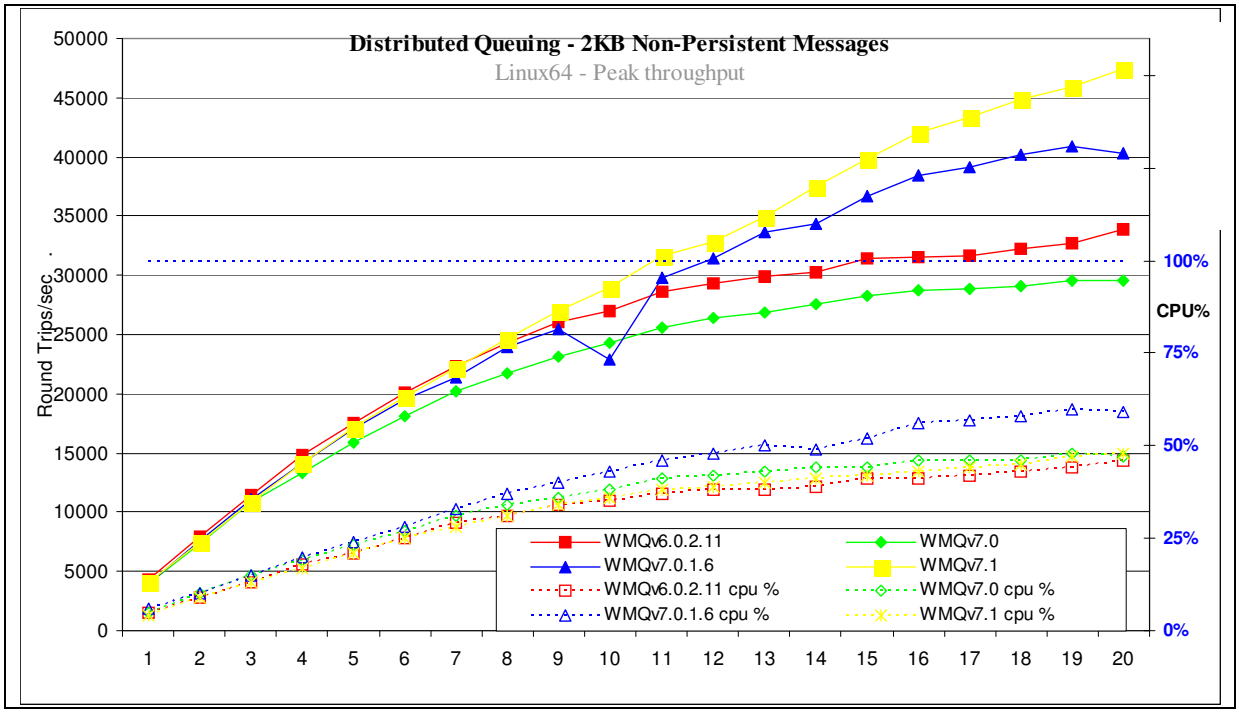


Figure 14 – Performance headline, non-persistent messages and server channels

Figure 14 and Table 10 shows that the throughput of non-persistent messages has increased by 16% when comparing version 7.1 to 7.0.1.6 and by 40% 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	20	33855	0.00091	46%
WMQv7.0	19	29587	0.00088	48%
WMQv7.0.1.6	19	40875	0.00057	60%
WMQv7.1	20	47414	0.00051	48%

Table 10 – 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

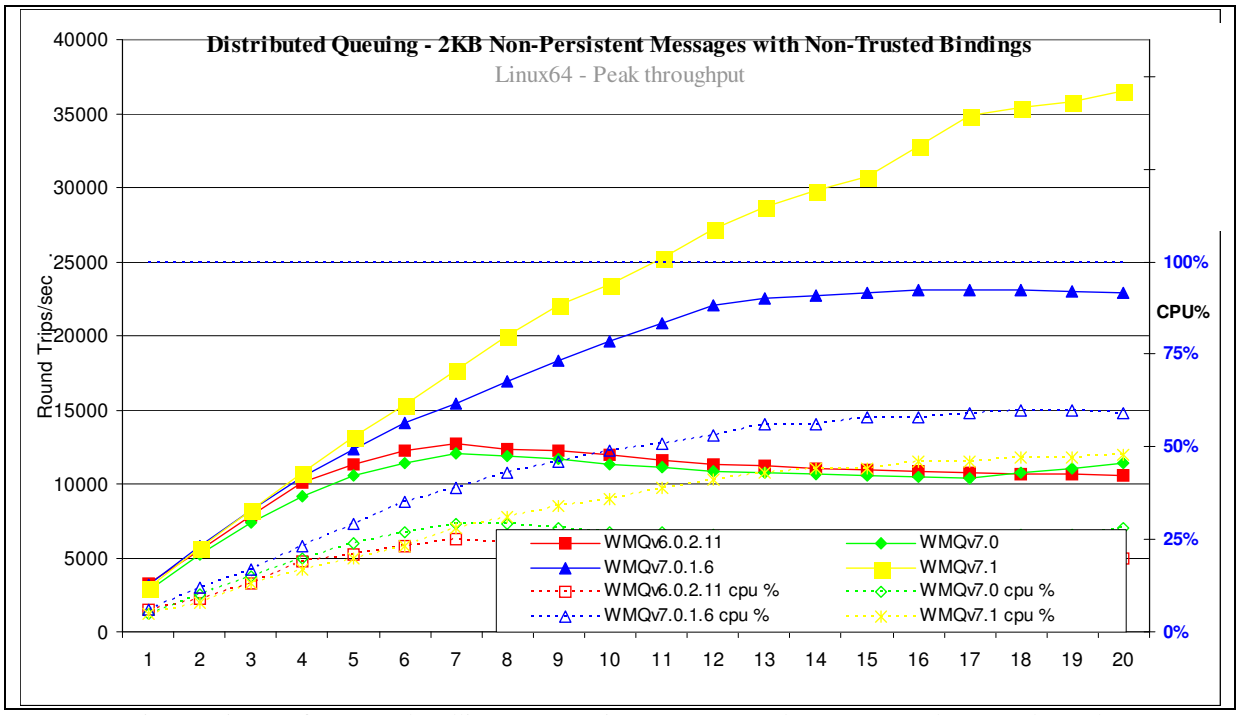


Figure 15 – Performance headline, non-persistent, not trusted messages and server channels

Figure 14 and Table 10 shows that the throughput Table 10 of non-persistent, non-trusted messages has increased by 58% when comparing version 7.1 to 7.0.1.6 and by 188% 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	7	12693	0.00064	25%
WMQv7.0	7	12026	0.00067	29%
WMQv7.0.1.6	17	23122	0.00088	59%
WMQv7.1	20	36515	0.00069	48%

Table 11 – 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

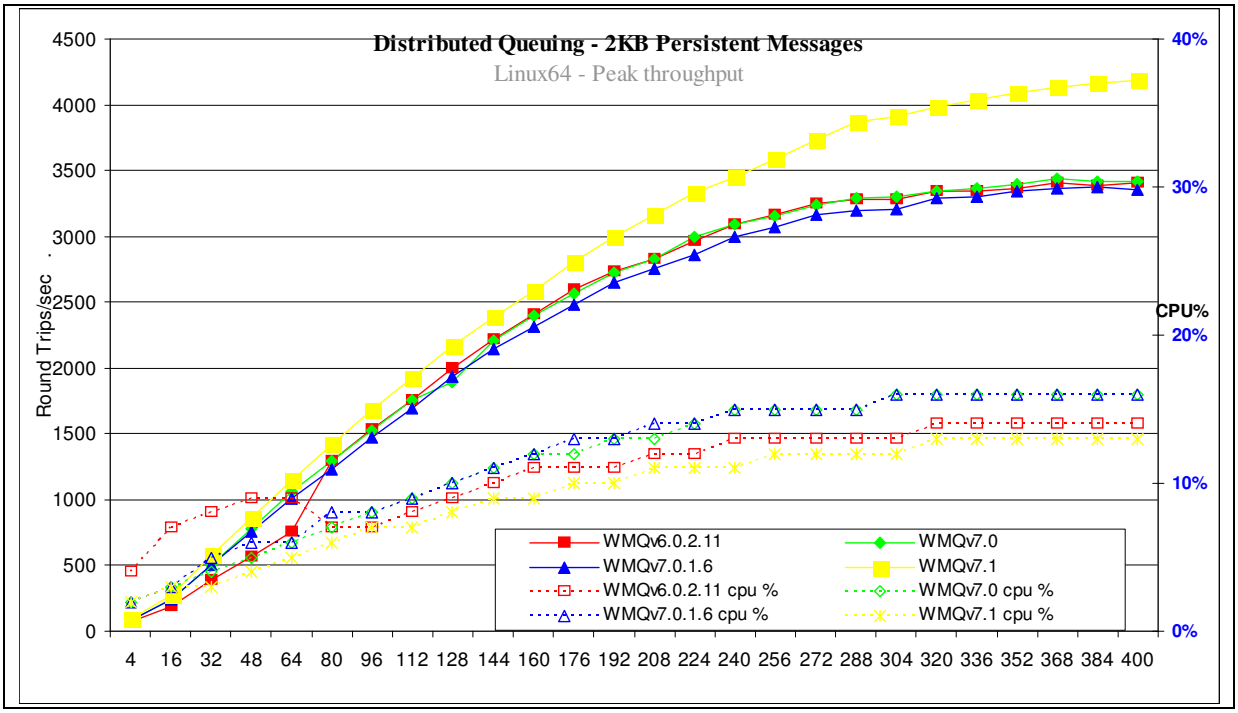


Figure 16 – Performance headline, persistent messages and server channels

Figure 16 and Table 12 shows that the throughput of persistent messages has increased by 24% when comparing version 7.1 to 7.0.1.6 and by 23% 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	368	3407	0.12	14%
WMQv7.0	368	3440	0.12	16%
WMQv7.0.1.6	384	3375	0.14	16%
WMQv7.1	400	4180	0.11	13%

Table 12 – 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.

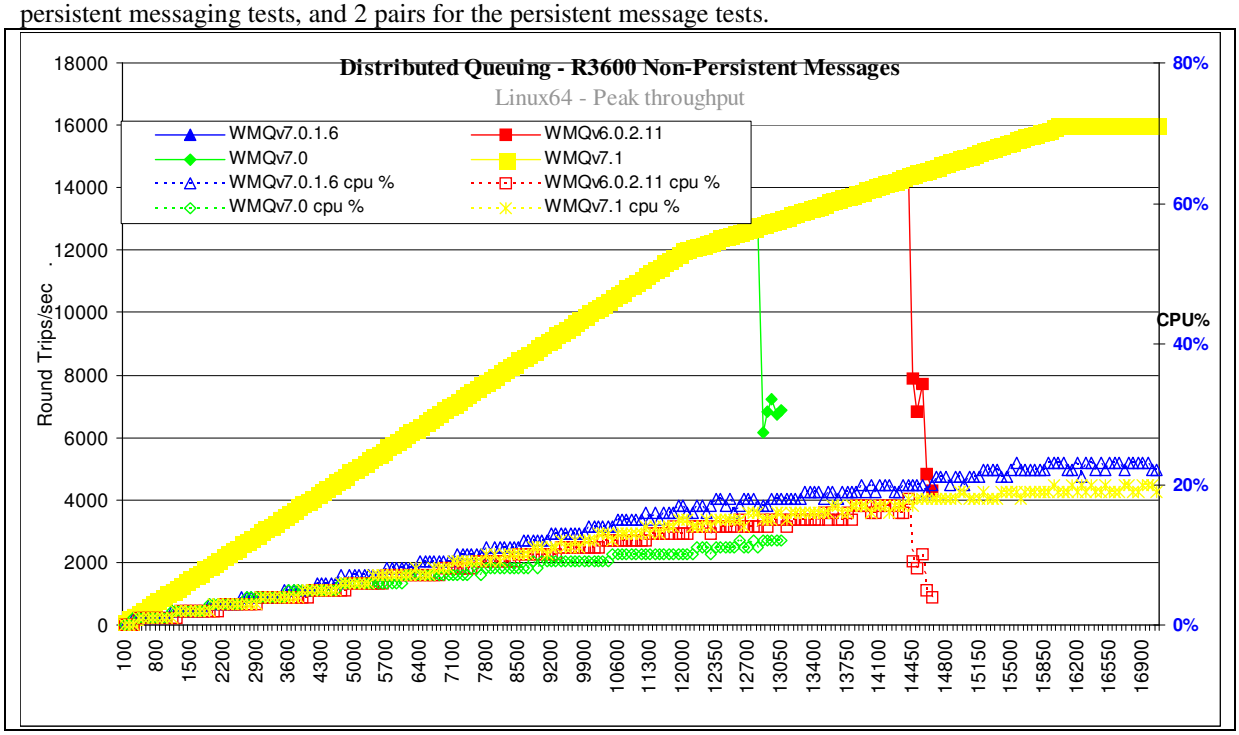


Figure 17 – 1 round trip per driving application per second, server channel, non-persistent messages

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

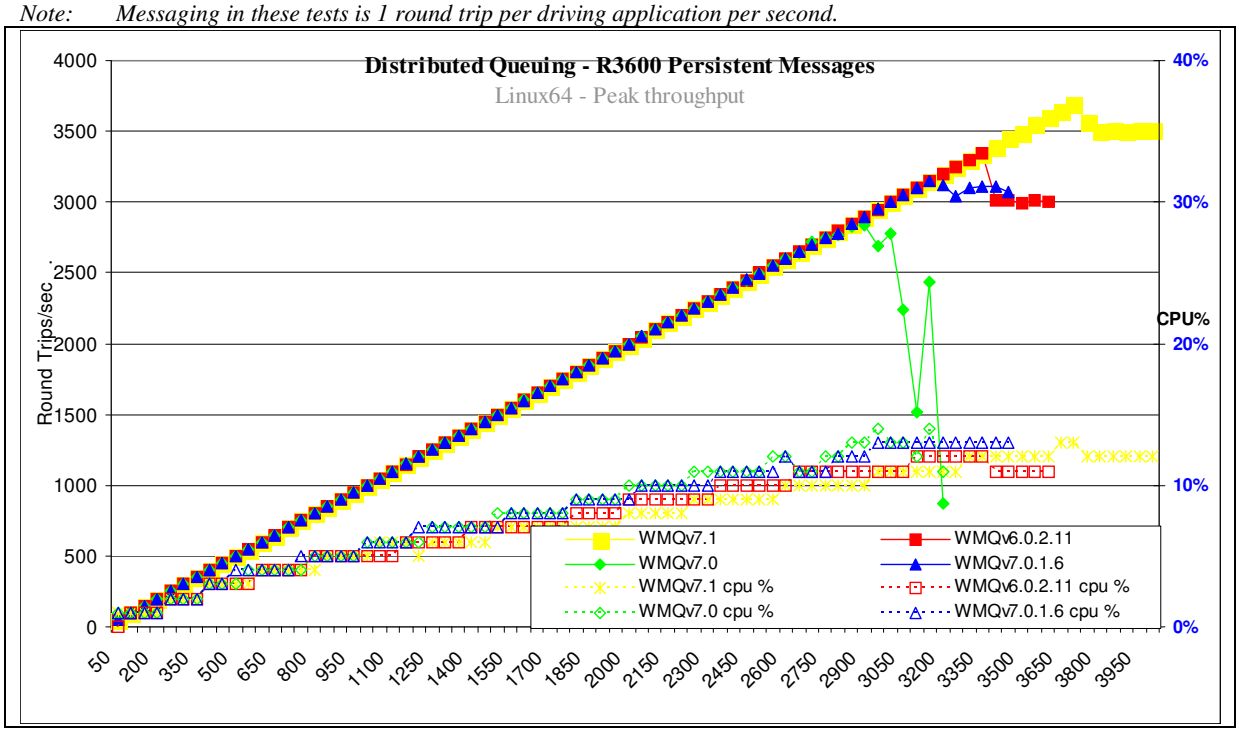


Figure 18 – 1 round trip per driving application per second, server channel, persistent messages

Figure 17, Figure 18 and Table 12 shows that the throughput of non-persistent messages is unchanged when comparing version 7.1 to 7.0.1.6 but has increased by 11% when comparing version 7.1 to 6.0.2.11 and for persistent messages has increased by 18% 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: Distributed Queuing - R3600 Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	14400	14395	0.22	18%
WMQv7.0	12800	12797	0.026	11%
WMQv7.0.1.6	16850	15995	0.00084	23%
WMQv7.1	16750	15995	0.00055	20%

Test Name: Distributed Queuing - R3600 Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	3350	3349	0.22	12%
WMQv7.0	2900	2834	0.49	13%
WMQv7.0.1.6	3150	3147	0.085	13%
WMQv7.1	3700	3699	0.1	13%

Table 13 – 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

2.3.5 SSL

The following diagram shows how throughput varies depending on the cipher selected. The top line (using the standard 2KB message DQNP test) in not encrypted. The other lines show a selection of the available ciphers.

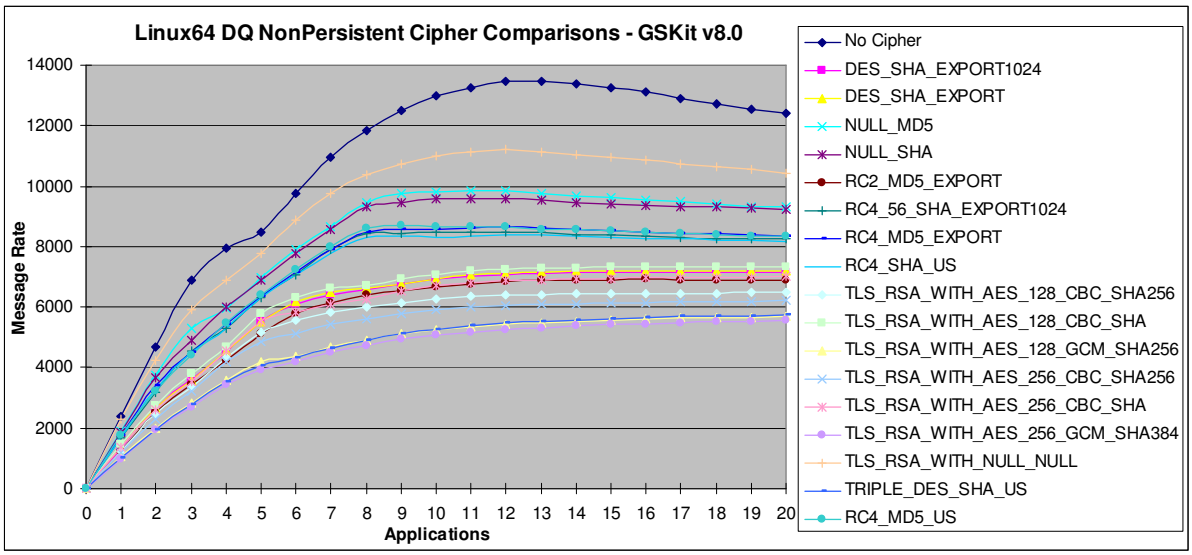


Figure 19 – DQ non-persistent message rates with various SSL ciphers

The area under the curve values for each cipher are used to compile an ordered table of these ciphers and can be used as a guide to expected performance degradations that can be expected when these ciphers are used. The unencrypted value is used to rate the other values as a percentage. Thus for the SSL cipher *TLS_RSA_WITH_NULL_NULL*, the expected message rate would be approximately 85% of the unencrypted rate.

Linux64 Cipher Comparisons	DQNP
No Cipher	100%
TLS_RSA_WITH_NULL_NULL	85%
NULL_MD5	76%
NULL_SHA	74%
RC4_MD5_US	67%
RC4_MD5_EXPORT	67%
RC4_56_SHA_EXPORT1024	66%
RC4_SHA_US	66%

Linux64 Cipher Comparisons	DQNP
TLS_RSA_WITH_AES_128_CBC_SHA	57%
DES_SHA_EXPORT	56%
DES_SHA_EXPORT1024	56%
TLS_RSA_WITH_AES_256_CBC_SHA	54%
RC2_MD5_EXPORT	53%
TLS_RSA_WITH_AES_128_CBC_SHA256	51%
TLS_RSA_WITH_AES_256_CBC_SHA256	48%
TRIPLE_DES_SHA_US	43%
TLS_RSA_WITH_AES_128_GCM_SHA256	43%
TLS_RSA_WITH_AES_256_GCM_SHA384	41%

Table 14 – Ordered relative SSL DQ cipher performance

3 Large Messages

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

3.1.1 50bytes to 32KB

Figure 20 show the response time for MQPut/MQGet for non-persistent message sizes between 50bytes and 32KB.

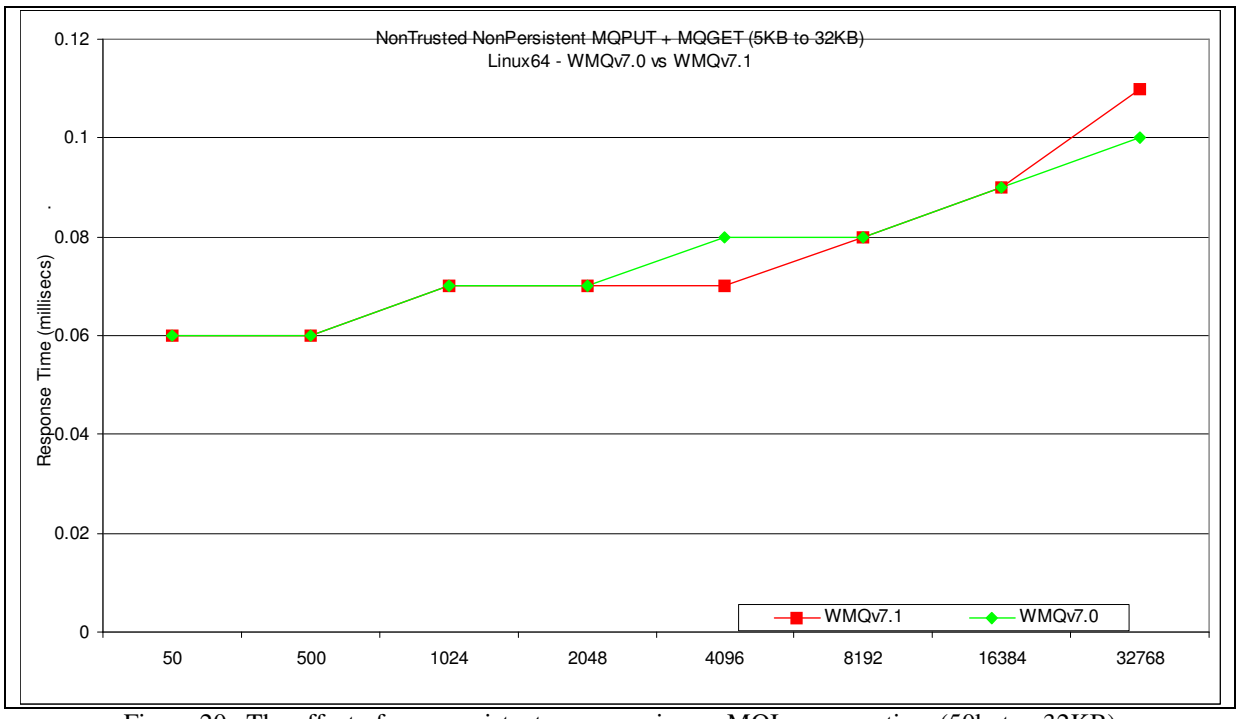


Figure 20 –The effect of non-persistent message size on MQI response time (50byte - 32KB)

Figure 21 show the response for MQPut/MQGet pairs for persistent message sizes between 50bytes and 32KB.

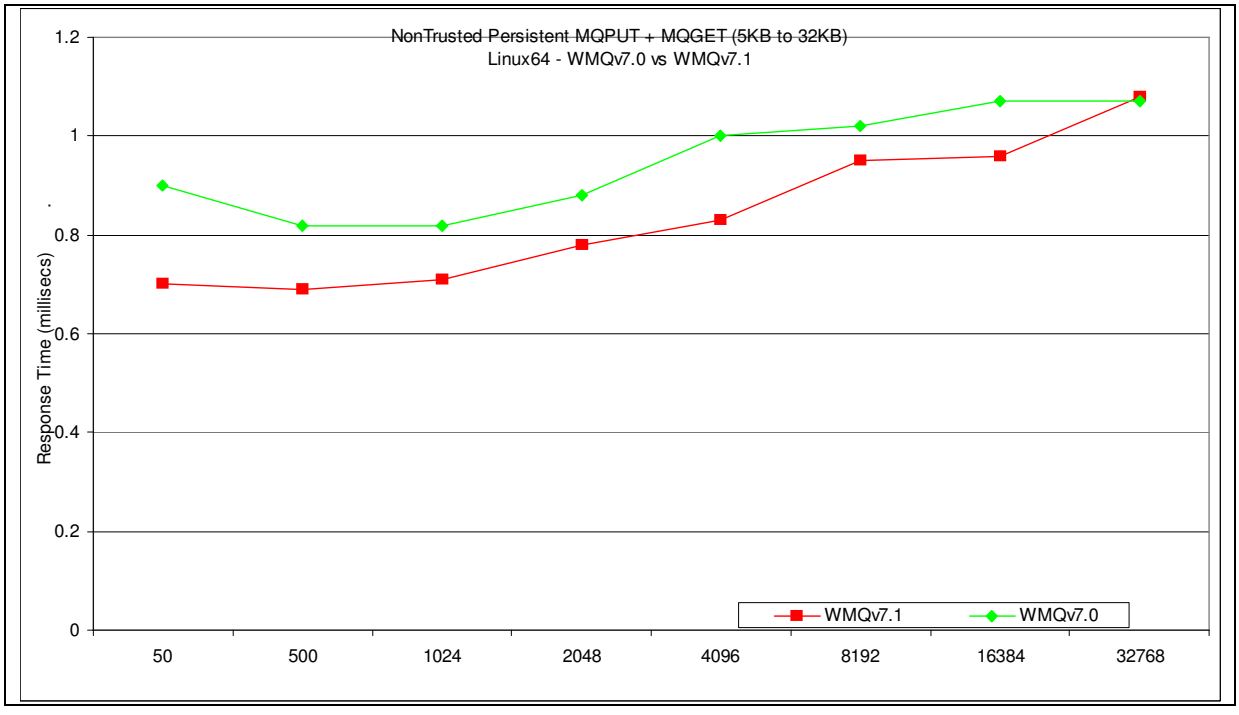


Figure 21 –The effect of persistent message size on MQI response time (50byte - 32KB)

3.1.2 32KB to 2MB

Figure 22 show the response time for MQPut/MQGet pairs has improved for all non-persistent message sizes between 32KB and 2MB.

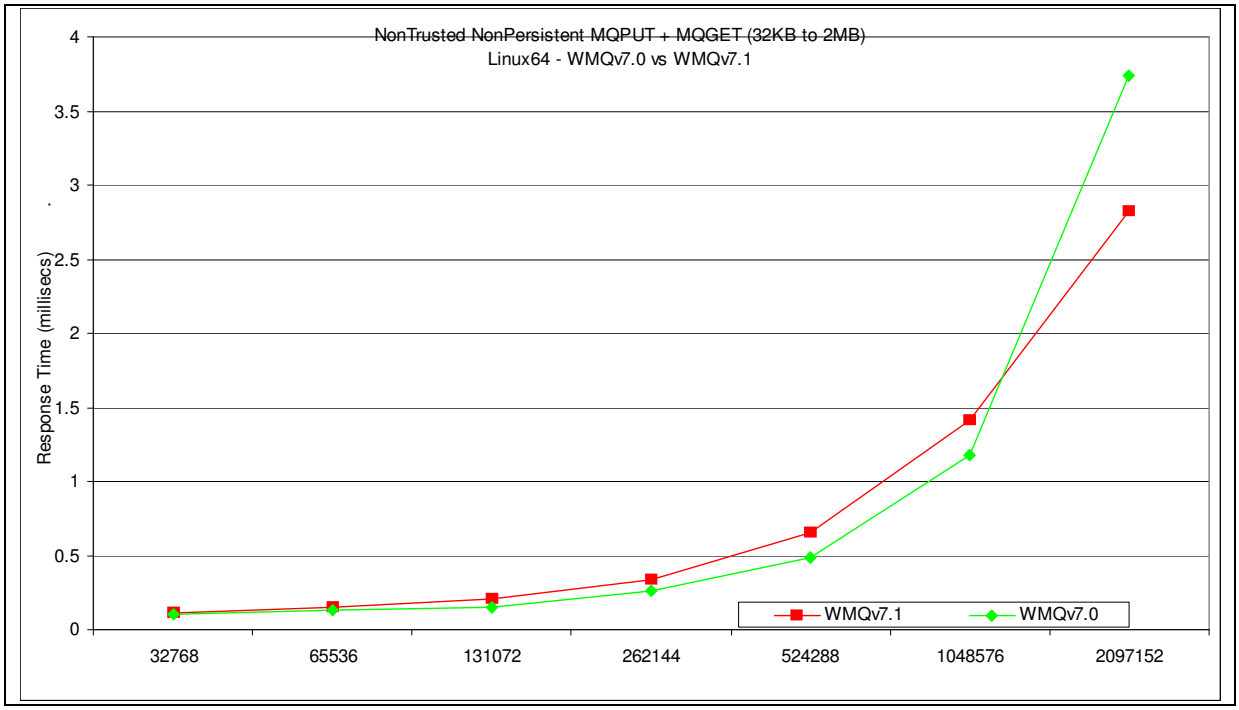


Figure 22 –The effect of non-persistent message size on MQI response time (32KB – 2MB)

Figure 23 show the response for MQPut/MQGet pairs for persistent message sizes between 32KB and 2MB.

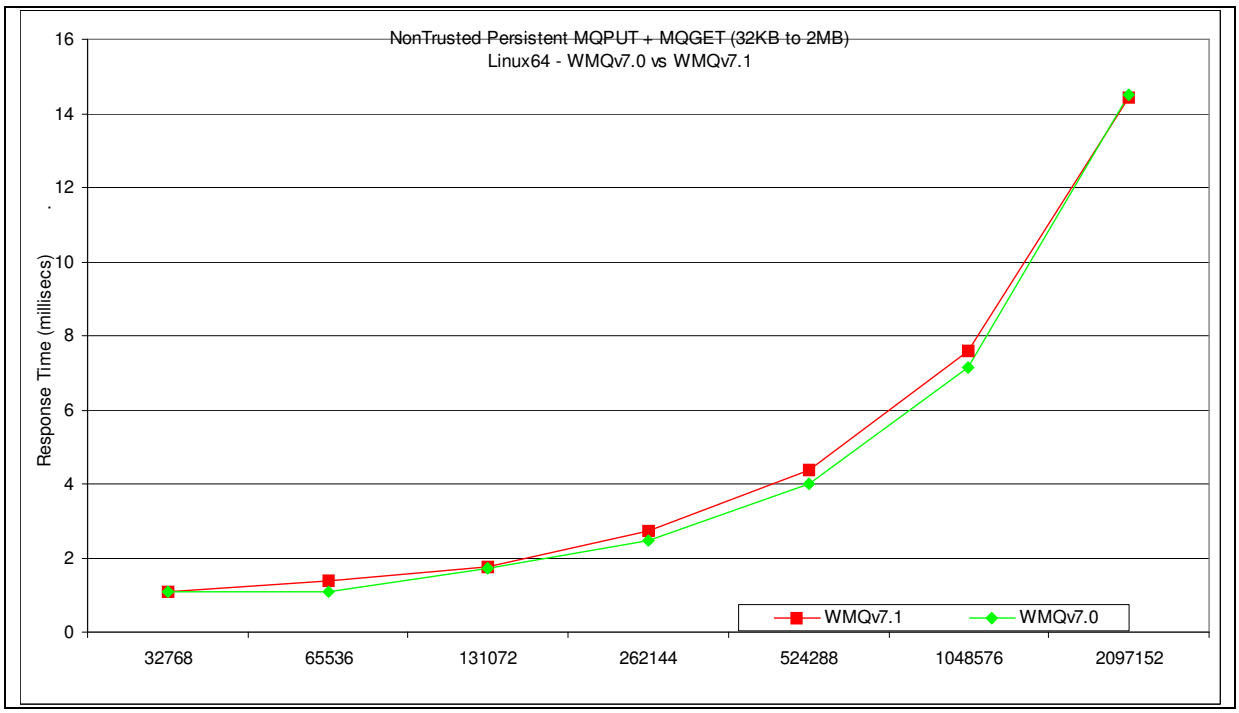


Figure 23 –The effect of persistent message size on MQI response time (32KB – 2MB)

3.1.3 2MB to 100MB

Figure 22 Response time for MQPut/MQGet pairs for NP message between 2MB and 100MB.

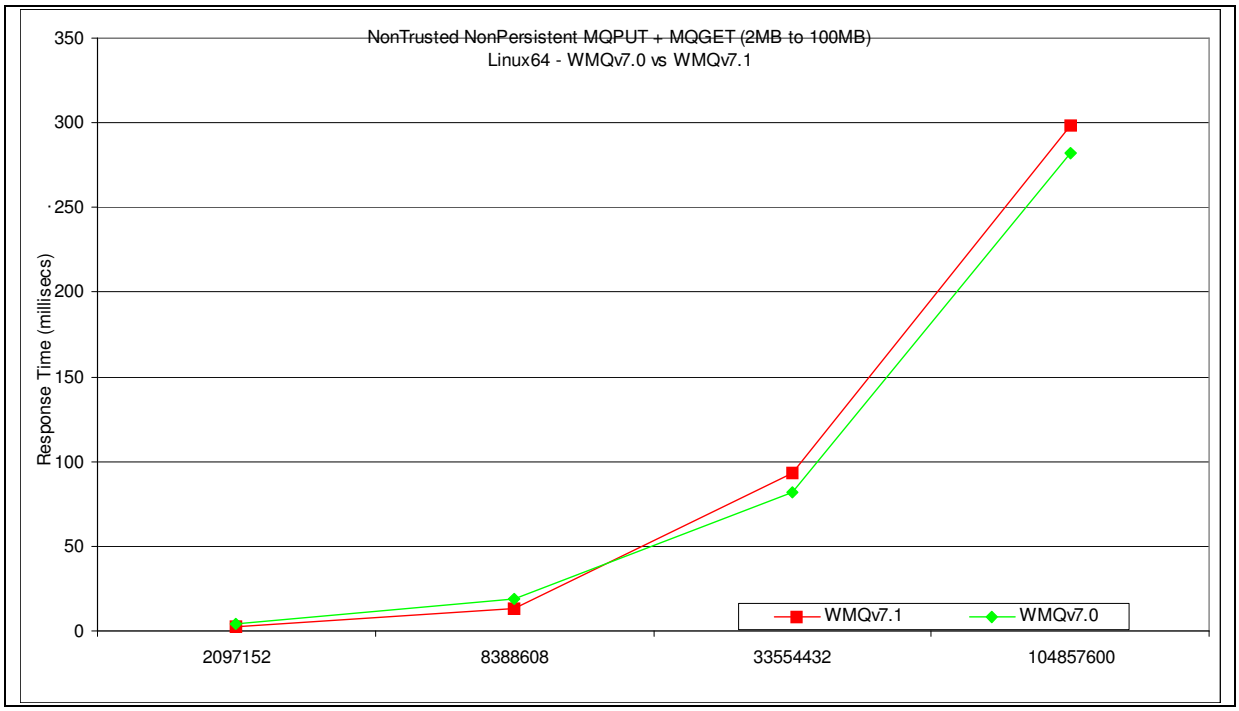


Figure 24 –The effect of non-persistent message size on MQI response time (2MB – 100MB)

Figure 25 The response for MQPut/MQGet pairs for persistent message sizes between 2MB and 32MB.

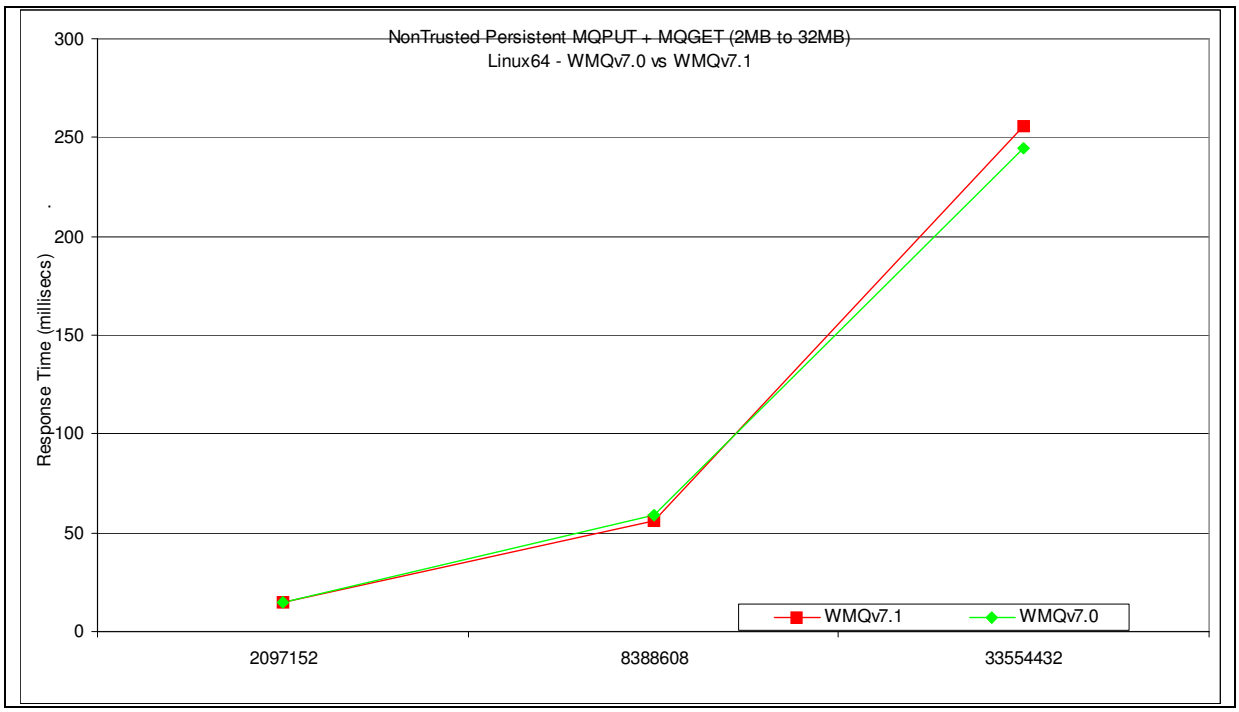


Figure 25 –The effect of persistent message size on MQI response time (2MB – 32MB)

3.2 20KB Messages

3.2.1 Local Queue Manager

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

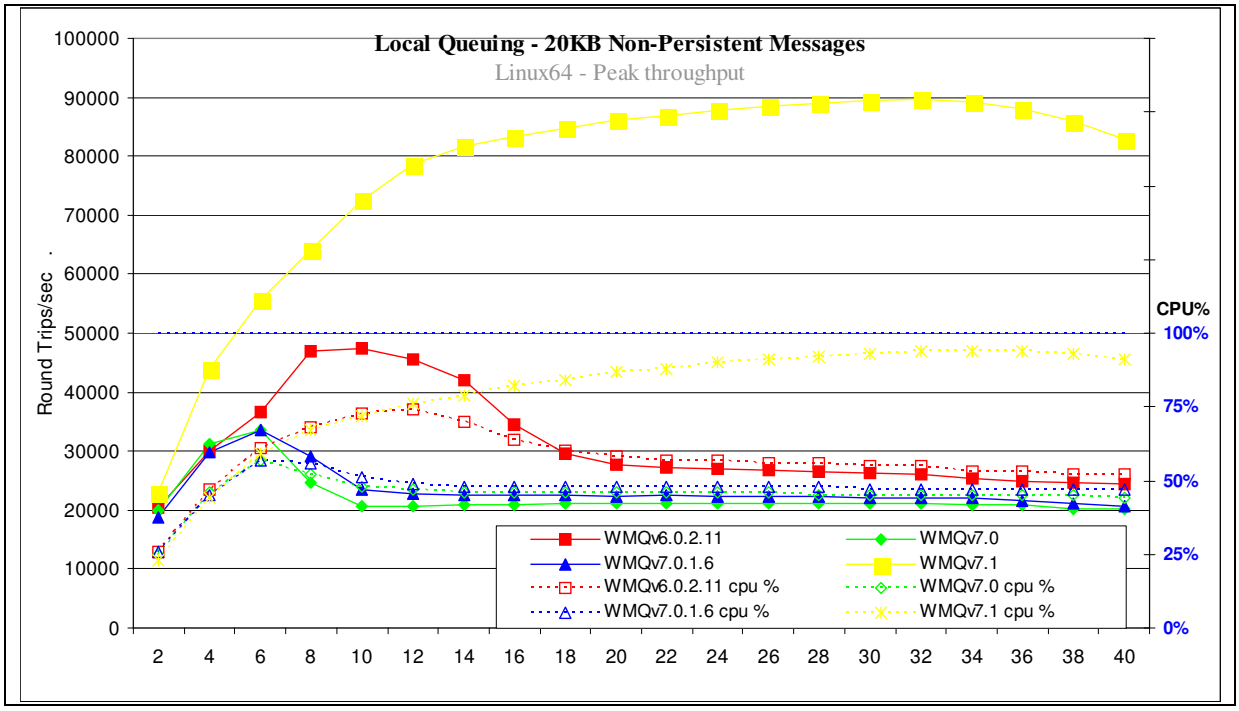


Figure 26 – 20KB non-persistent messages, local queue manager

Figure 26 and Table 15 shows that the throughput of non-persistent messages has increased by 168% when comparing version 7.1 to 7.0.1.6 and by 89% 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	10	47330	0.00024	73%
WMQv7.0	6	33651	0.0002	57%
WMQv7.0.1.6	6	33513	0.00018	57%
WMQv7.1	32	89670	0.0004	94%

Table 15 – 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

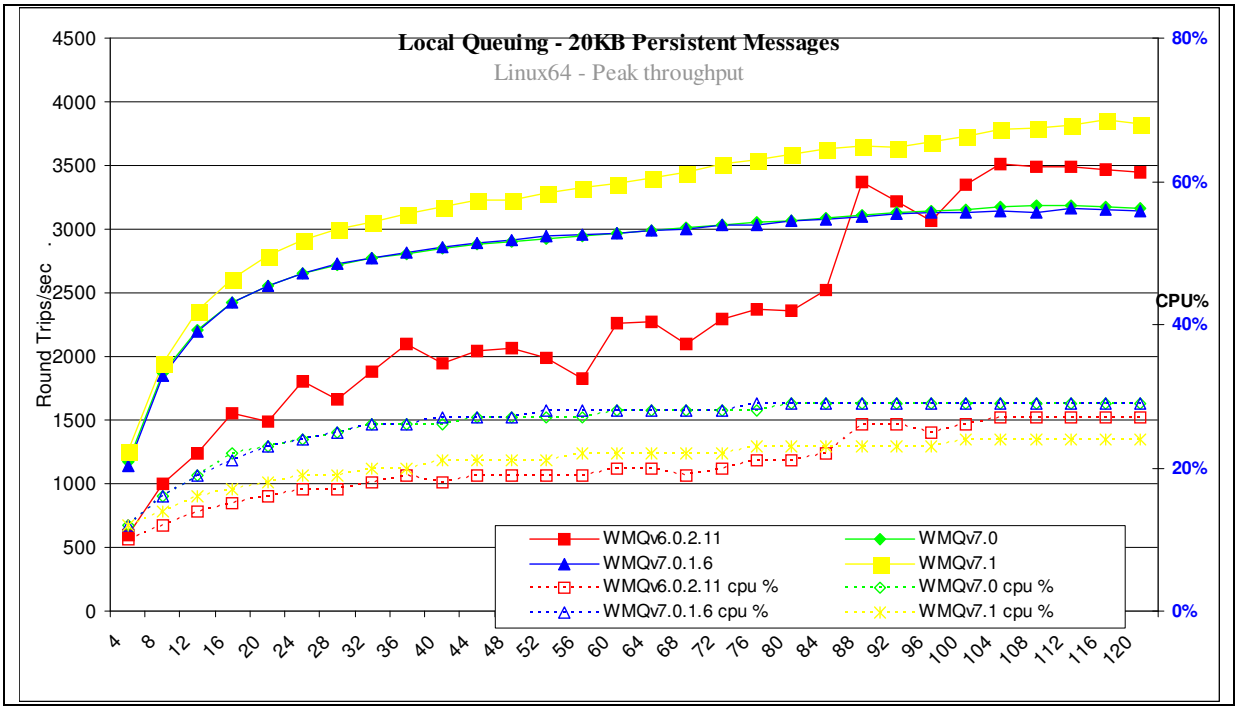


Figure 27 – 20KB persistent messages, local queue manager

Figure 27 and Table 16 shows that the throughput of persistent messages has increased by 22% 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: Local Queuing - 20KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	104	3507	0.034	27%
WMQv7.0	112	3180	0.041	29%
WMQv7.0.1.6	112	3161	0.042	29%
WMQv7.1	116	3853	0.035	24%

Table 16 – 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

3.2.1.2 Scalability Local 20K non Persistent

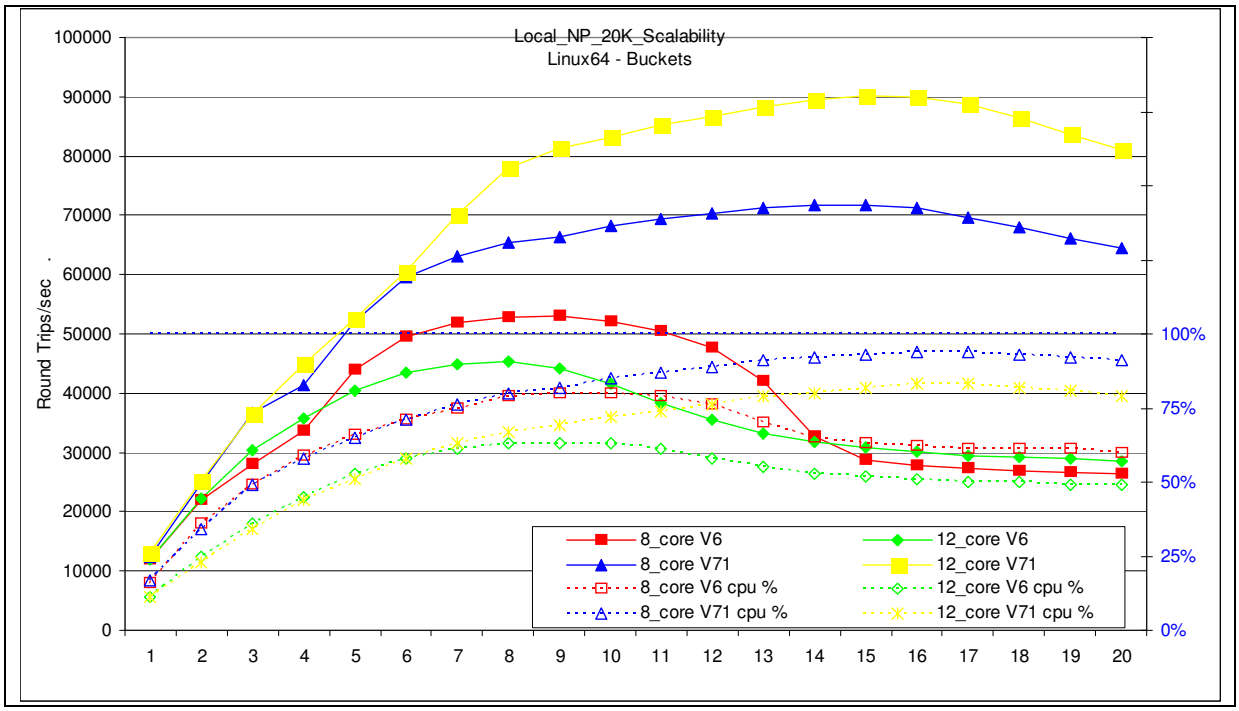


Figure 28 – Scalability, Local Queue manager, non-persistent messages

Test Name: Local_NP_20K_Scalability	Apps	Round Trips/Sec	Response time (s)	CPU
8_core V6	9	52930	0.0002	80%
12_core V6	8	45362	0.00023	63%
8_core V71	14	71718	0.00022	92%
12_core V71	15	90254	0.0002	82%

Table 17 – Scalability, Local Queue manager, non-persistent messages

The throughput comparison of 8 core and 12 cores for MQ V 6.0.2.10 messages shows better throughput can be achieved with 8 cores. The throughput comparison for MQ V7.1 with 12 cores is 19% larger with the peak difference being 25% larger. The significant improvement over previous levels is maximised in scenarios where messages all go through a single queue.

3.2.2 Client Channel

Figure 27 and Figure 28 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

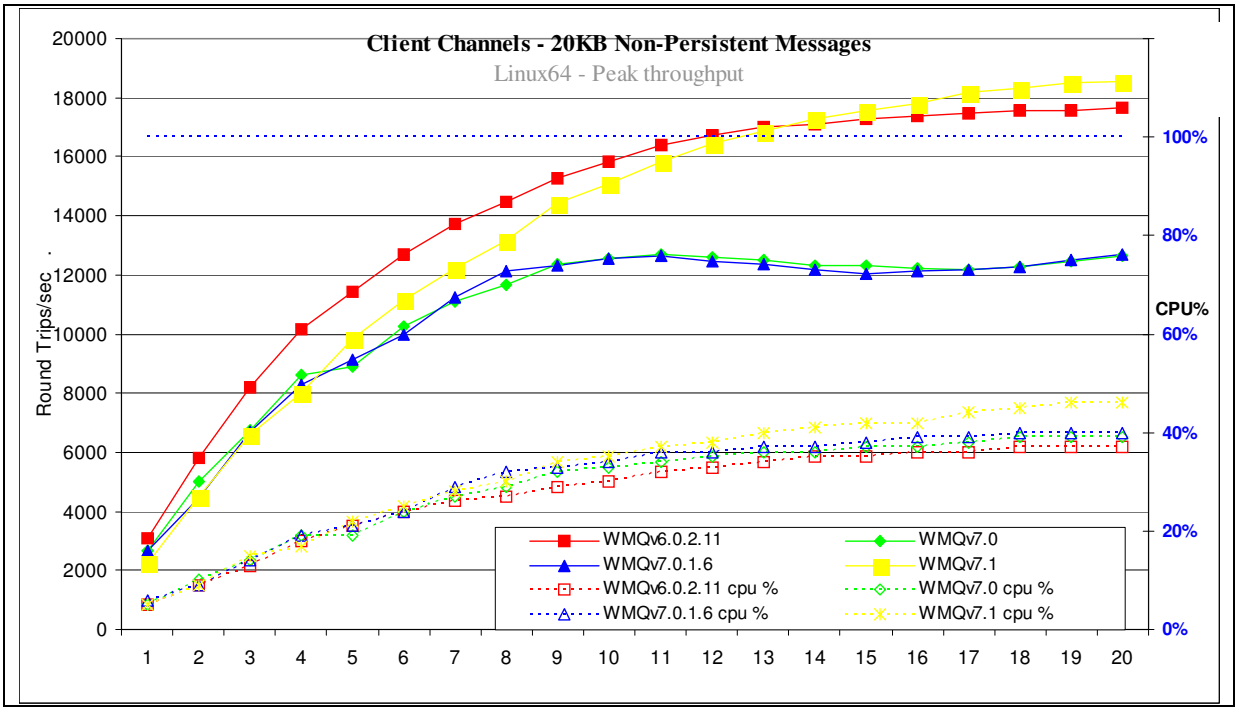


Figure 29 – 20KB non-persistent messages, client channels

Figure 27 and Table 18 shows that the throughput of non-persistent messages has increased by 46% when comparing version 7.1 to 7.0.1.6 and by 5% when comparing version 7.1 to 6.0.2.11.

Test Name: Client Channels - 20KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	20	17645	0.0013	37%
WMQv7.0	11	12713	0.001	34%
WMQv7.0.1.6	20	12704	0.0017	40%
WMQv7.1	20	18534	0.0012	46%

Table 18 – 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

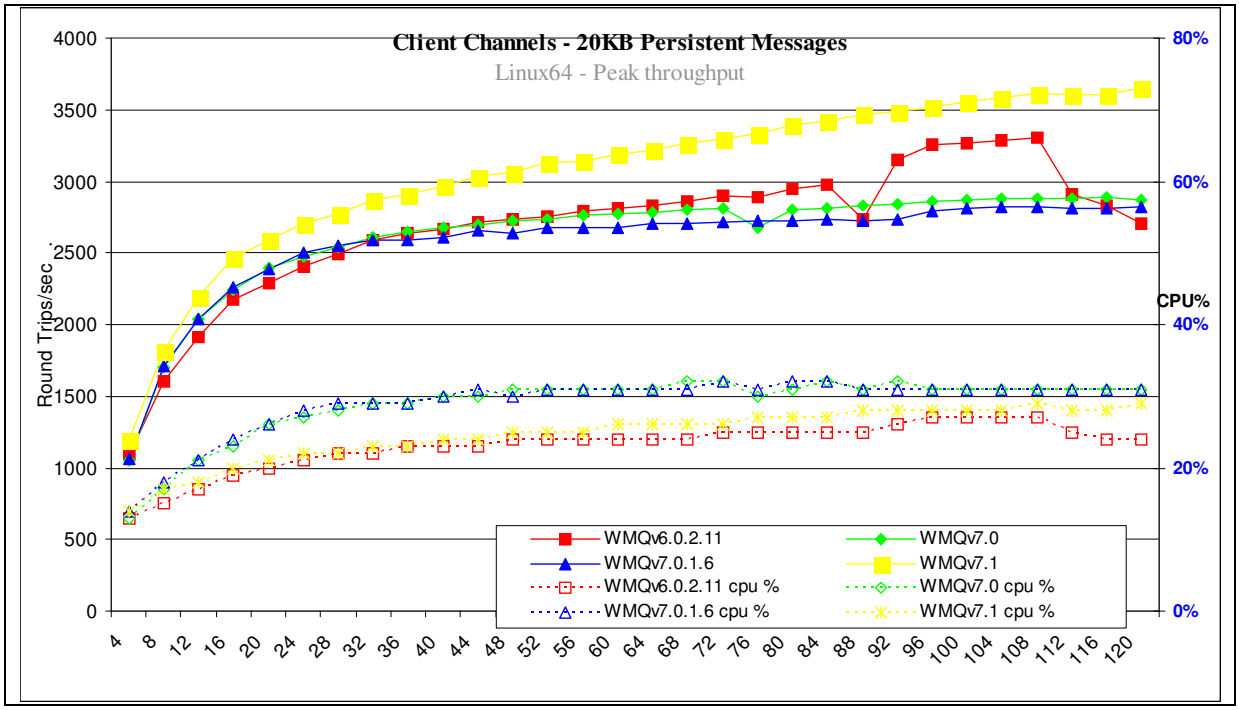


Figure 30 – 20KB persistent messages, client channels

Figure 28 and Table 19 shows that the throughput of persistent messages has increased by 29% 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 - 20KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	108	3306	0.037	27%
WMQv7.0	116	2886	0.047	31%
WMQv7.0.1.6	104	2824	0.044	31%
WMQv7.1	120	3652	0.037	29%

Table 19 – 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 31 and Figure 30 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

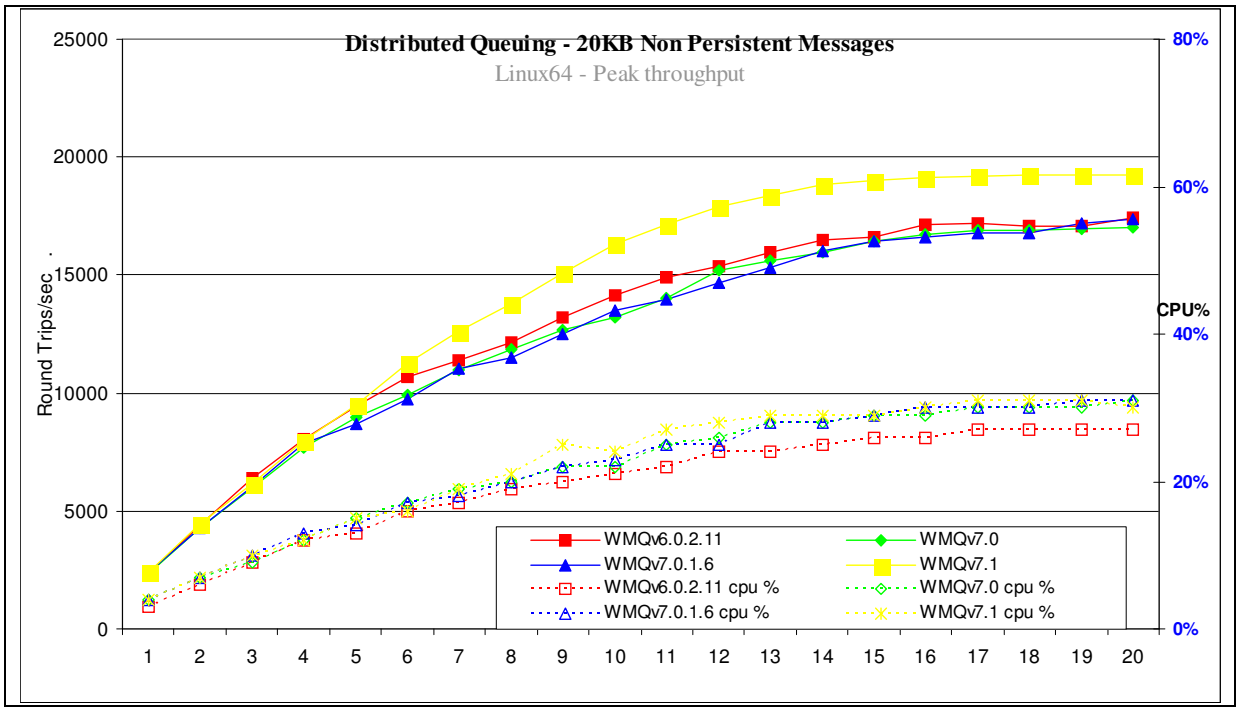


Figure 31 – 20KB non-persistent messages, distributed queuing

Figure 31 and Table 20 shows that the throughput of non-persistent messages has increased by 11% when comparing version 7.1 to 7.0.1.6 and 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	20	17419	0.0013	27%
WMQv7.0	20	17014	0.0013	31%
WMQv7.0.1.6	20	17395	0.0013	31%
WMQv7.1	20	19257	0.0012	30%

Table 20 – 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

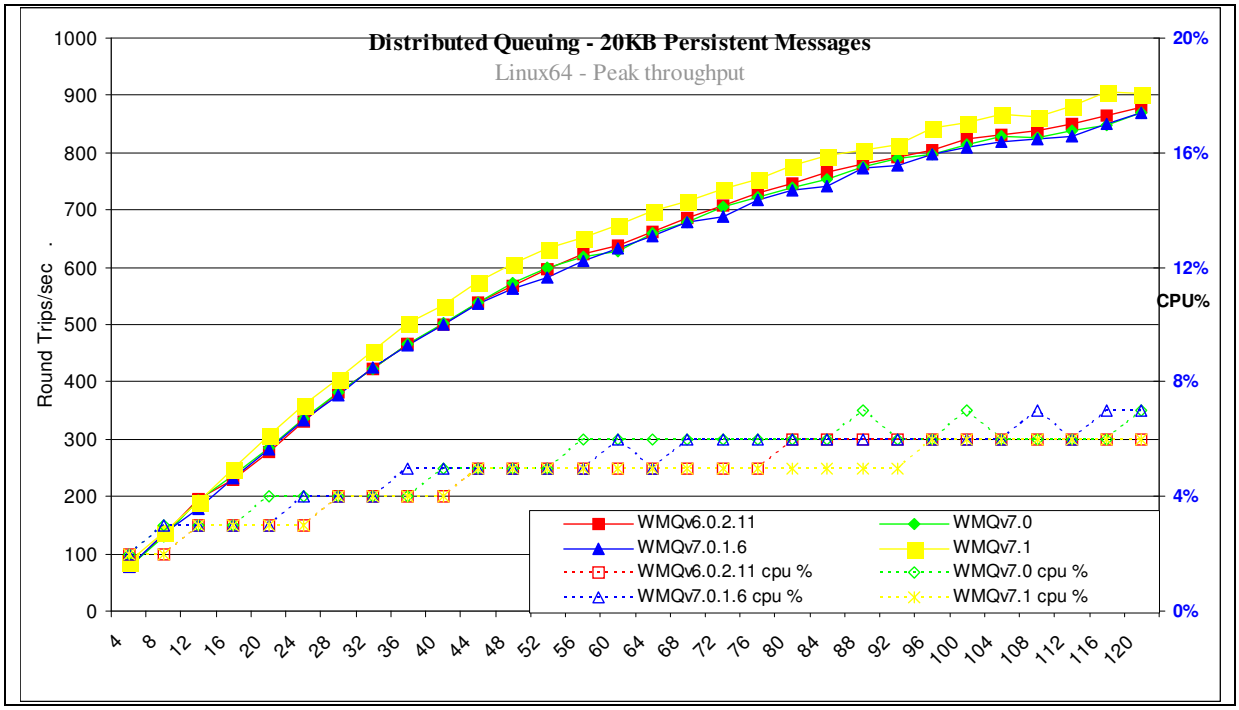


Figure 32 – 20KB persistent messages, distributed queuing

Figure 30 and Table 21 shows that the throughput of persistent messages has not changed when comparing version 7.1 to 7.0.1.6 and to 6.0.2.11.

Test Name: Distributed Queuing - 20KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	120	879	0.16	6%
WMQv7.0	120	870	0.15	7%
WMQv7.0.1.6	120	868	0.16	7%
WMQv7.1	116	905	0.14	6%

Table 21 – 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 31 and Figure 32 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

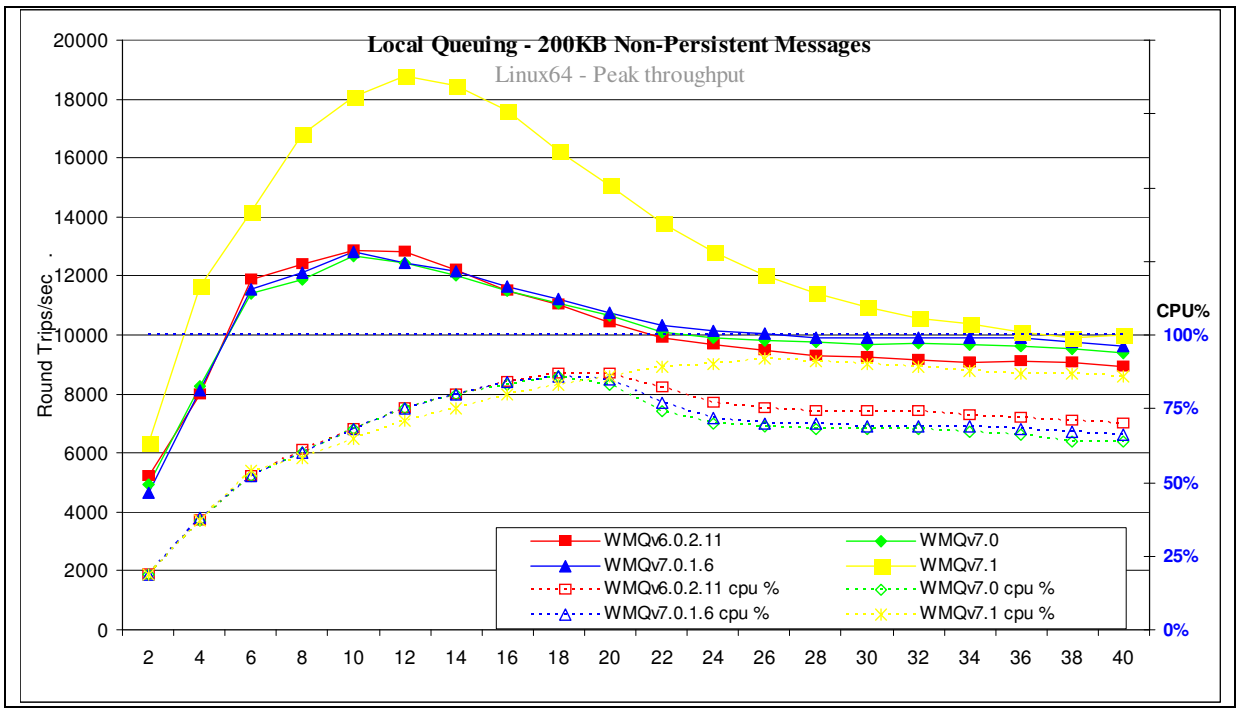


Figure 33 – 200KB non-persistent messages, local queue manager

Figure 31 and Table 22 shows that the throughput of non-persistent messages has increased by 47% when comparing version 7.1 to 7.0.1.6 and by 46% 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	10	12850	0.00085	68%
WMQv7.0	10	12699	0.0009	68%
WMQv7.0.1.6	10	12806	0.00091	68%
WMQv7.1	12	18802	0.0007	71%

Table 22 – 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

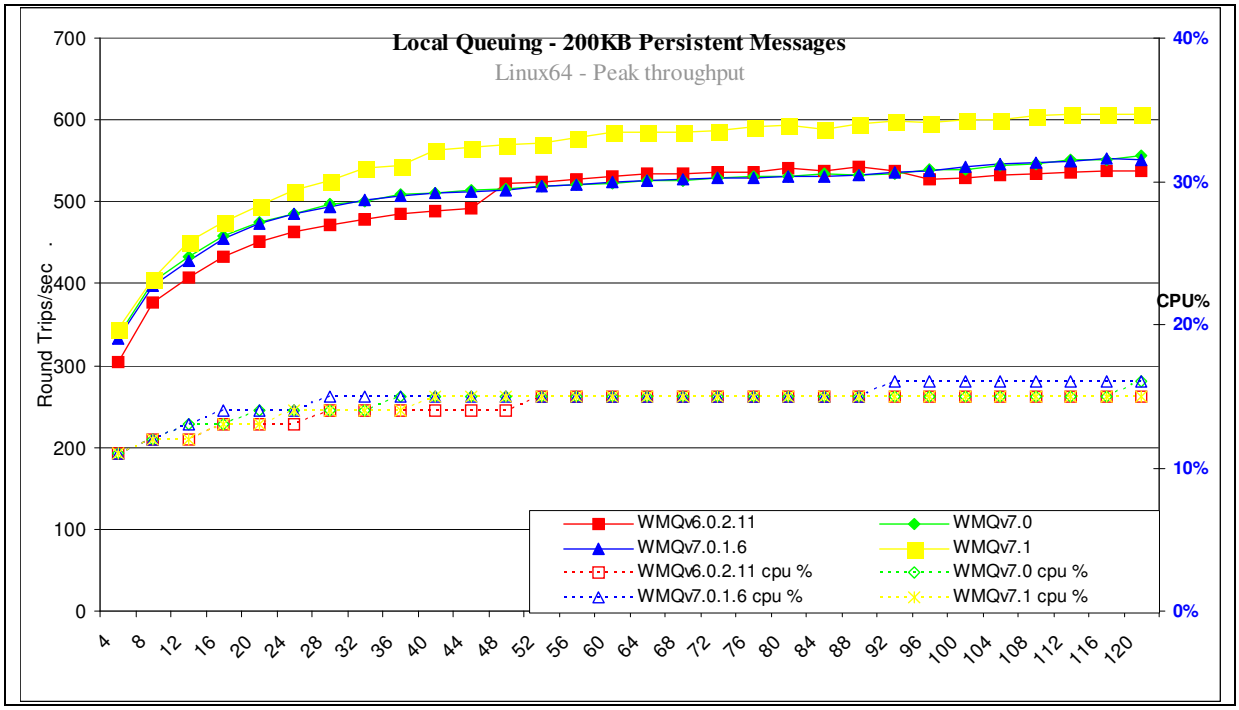


Figure 34 – 200KB persistent messages, local queue manager

Figure 32 and Table 23 shows that the throughput of persistent messages has increased by 10% when comparing version 7.1 to 7.0.1.6 and by 12% when comparing version 7.1 to 6.0.2.11.

Test Name: Local Queuing - 200KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	88	542	0.19	15%
WMQv7.0	120	556	0.25	16%
WMQv7.0.1.6	116	553	0.24	16%
WMQv7.1	112	607	0.22	15%

Table 23 – 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 33 and Figure 34 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

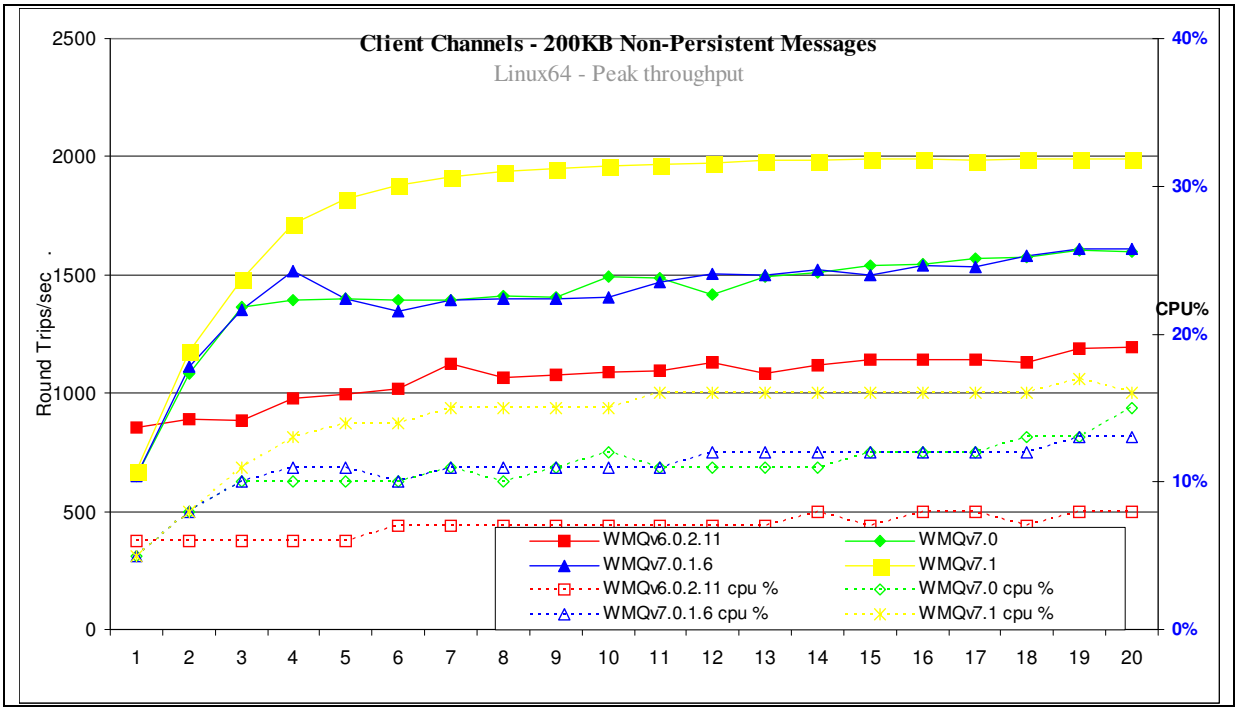


Figure 35 – 200KB non-persistent messages, client channels

Figure 33 and Table 24 shows that the throughput of non-persistent messages has increased by 24% when comparing version 7.1 to 7.0.1.6 and by 67% when comparing version 7.1 to 6.0.2.11.

Test Name: Client Channels - 200KB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	20	1194	0.018	8%
WMQv7.0	19	1604	0.014	13%
WMQv7.0.1.6	20	1609	0.016	13%
WMQv7.1	18	1993	0.01	16%

Table 24 – 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

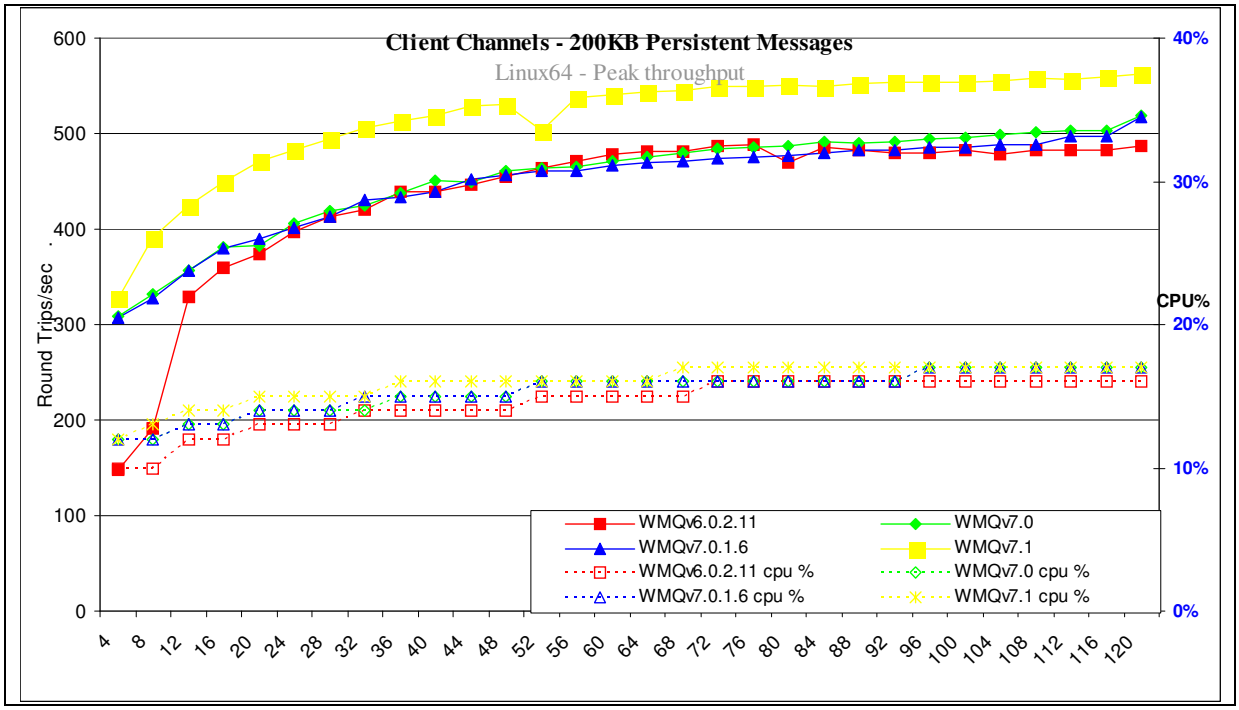


Figure 36 – 200KB persistent messages, client channels

Figure 34 and Table 25 shows that the throughput of persistent messages has increased by 9% when comparing version 7.1 to 7.0.1.6 and by 15% 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	76	488	0.19	16%
WMQv7.0	120	519	0.27	17%
WMQv7.0.1.6	120	517	0.27	17%
WMQv7.1	120	562	0.25	17%

Table 25 – 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 35 and Figure 36 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

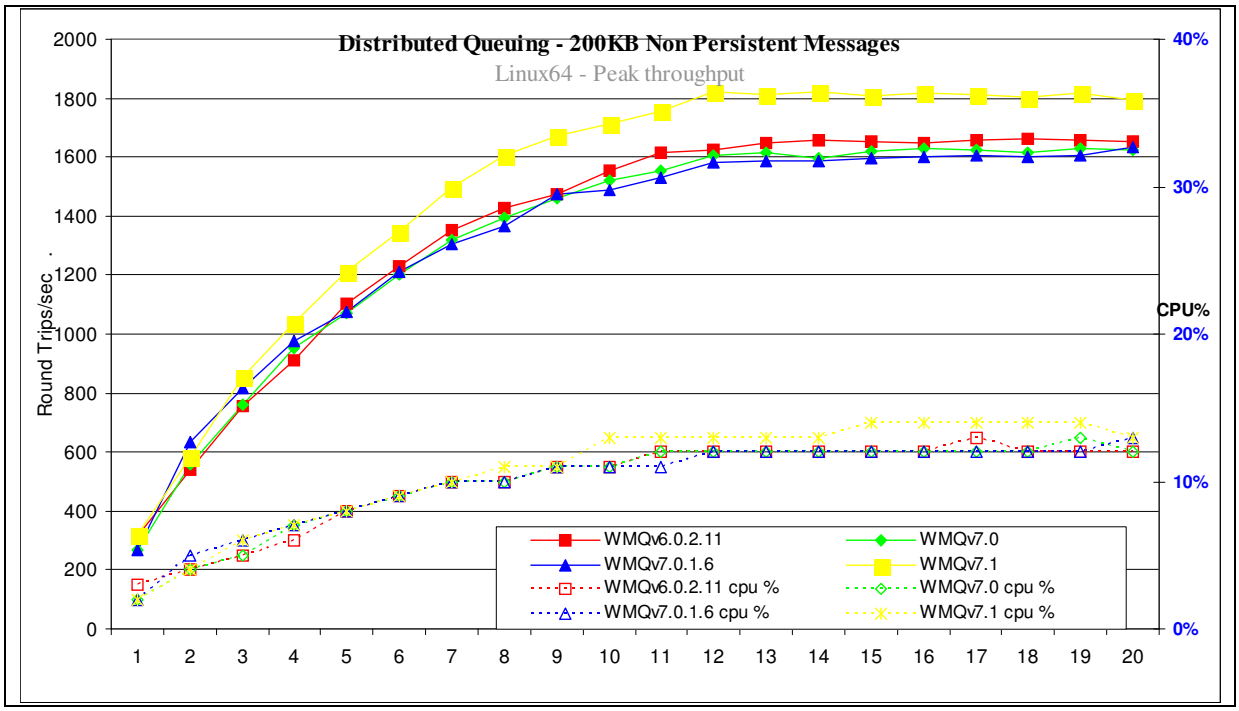


Figure 37 – 200KB non-persistent messages, distributed queuing

Figure 35 and Table 26 shows that the throughput of non-persistent messages has increased by 11% 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: Distributed Queuing - 200KB Non Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	18	1664	0.012	12%
WMQv7.0	19	1628	0.013	13%
WMQv7.0.1.6	20	1635	0.013	13%
WMQv7.1	14	1822	0.0088	13%

Table 26 – 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

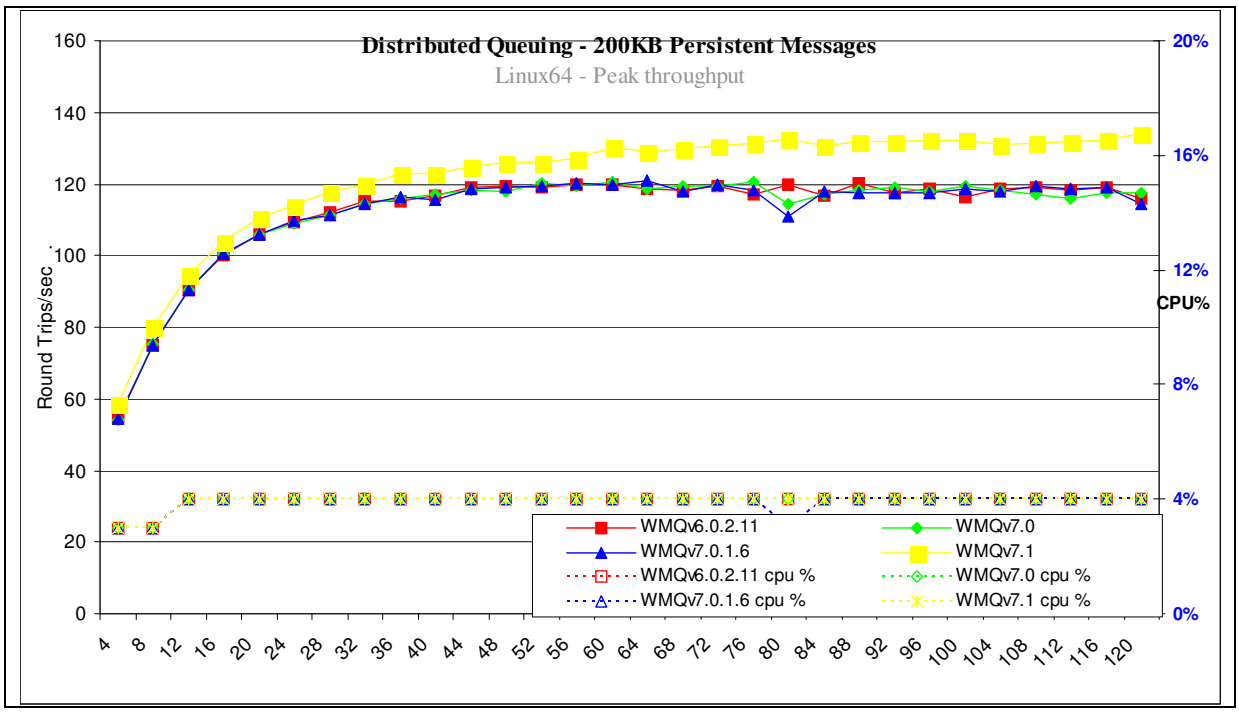


Figure 38 – 200KB persistent messages, distributed queuing

Figure 36 and Table 27 shows that the throughput of persistent messages has increased by 10% when comparing version 7.1 to 7.0.1.6 and to 6.0.2.11.

Test Name: Distributed Queuing - 200KB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	88	120	0.88	4%
WMQv7.0	60	120	0.61	4%
WMQv7.0.1.6	64	121	0.64	4%
WMQv7.1	80	133	0.67	4%

Table 27 – 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 37 and Figure 38 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

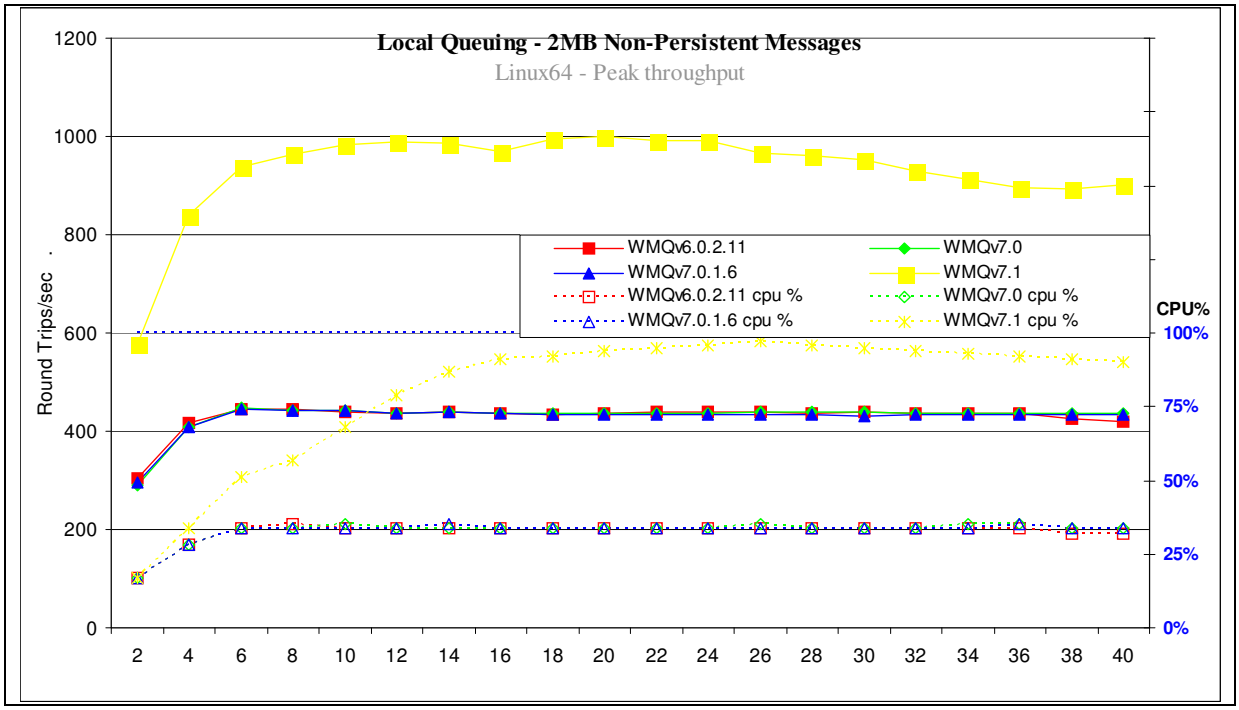


Figure 39 – 2MB non-persistent messages, local queue manager

Figure 37 and Table 28 shows that the throughput of non-persistent messages has increased by 124% when comparing version 7.1 to 7.0.1.6 and 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	6	445	0.014	34%
WMQv7.0	6	447	0.014	34%
WMQv7.0.1.6	6	446	0.014	34%
WMQv7.1	20	999	0.022	94%

Table 28 – 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

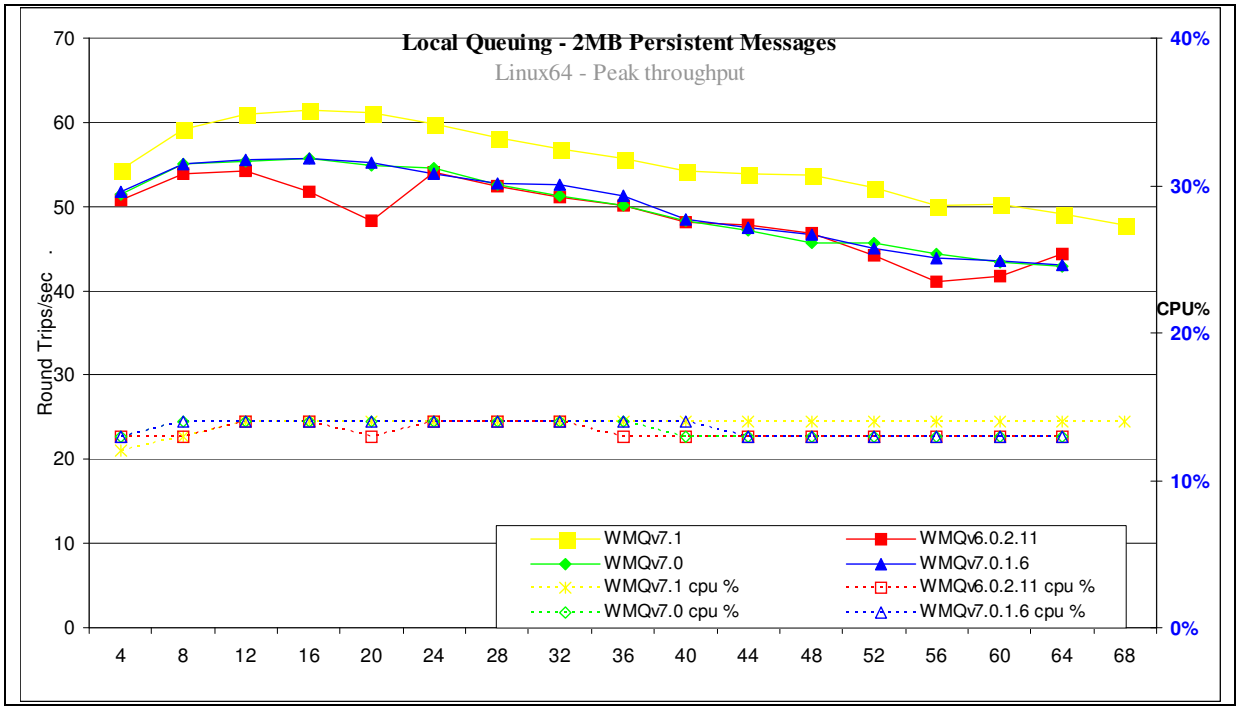


Figure 40 – 2MB persistent messages, local queue manager

Figure 38 and Table 29 shows that the throughput of persistent messages has increased by 10% 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: Local Queuing - 2MB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	12	54	0.25	14%
WMQv7.0	16	56	0.32	14%
WMQv7.0.1.6	16	56	0.32	14%
WMQv7.1	16	62	0.3	14%

Table 29 – 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 41 and Figure 40 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

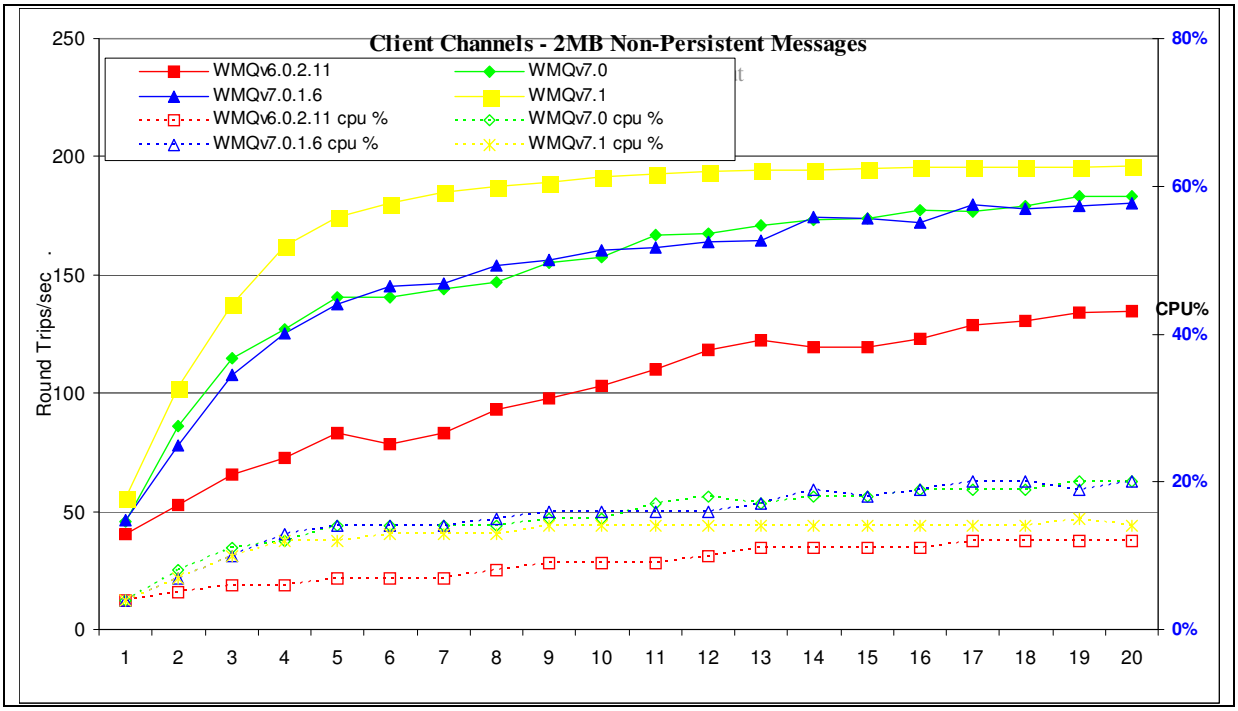


Figure 41 – 2MB non-persistent messages, client channels

Figure 41 and Table 30 shows that the throughput of non-persistent messages has increased by 9% when comparing version 7.1 to 7.0.1.6 and by 46% when comparing version 7.1 to 6.0.2.11.

Test Name: Client Channels - 2MB Non-Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	20	135	0.16	12%
WMQv7.0	20	184	0.11	20%
WMQv7.0.1.6	20	180	0.13	20%
WMQv7.1	20	196	0.11	14%

Table 30 – 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

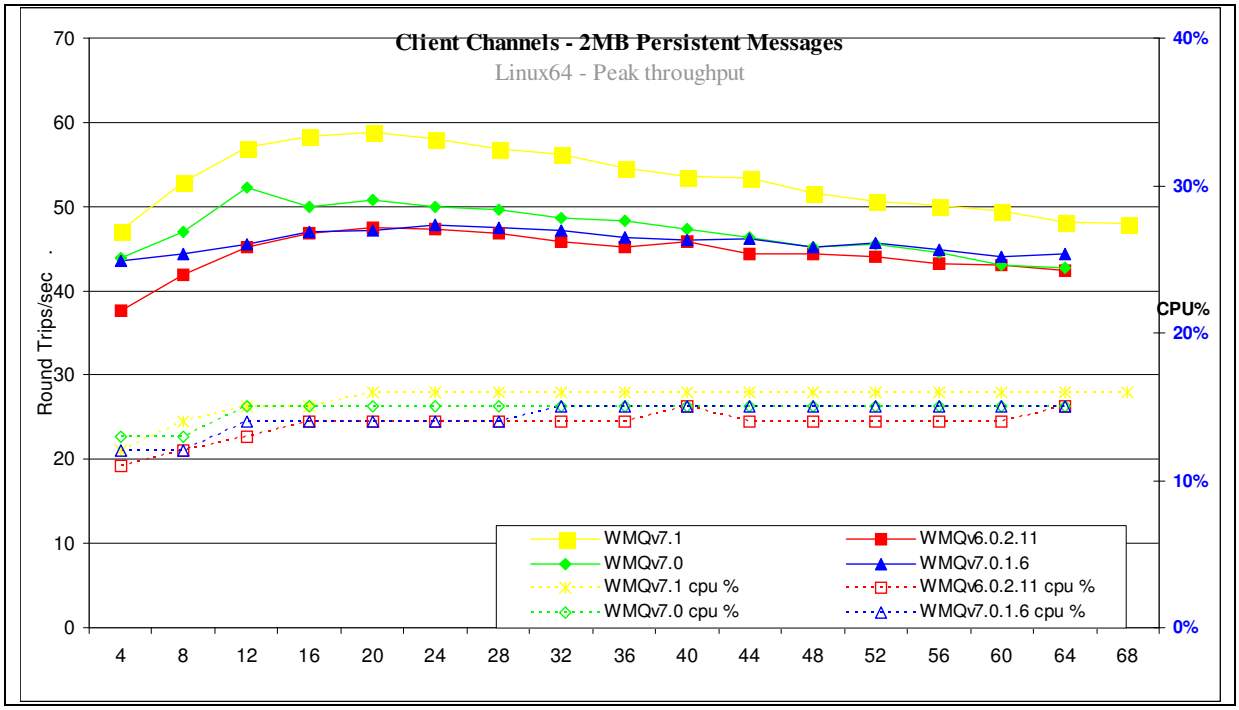


Figure 42 – 2MB persistent messages, client channels

Figure 40 and Table 31 shows that the throughput of persistent messages has increased by 23% when comparing version 7.1 to 7.0.1.6 and by 24% when comparing version 7.1 to 6.0.2.11.

Test Name: Client Channels - 2MB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	20	47	0.41	14%
WMQv7.0	12	52	0.24	15%
WMQv7.0.1.6	24	48	0.47	14%
WMQv7.1	20	59	0.39	16%

Table 31 – 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 41 and Figure 42 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

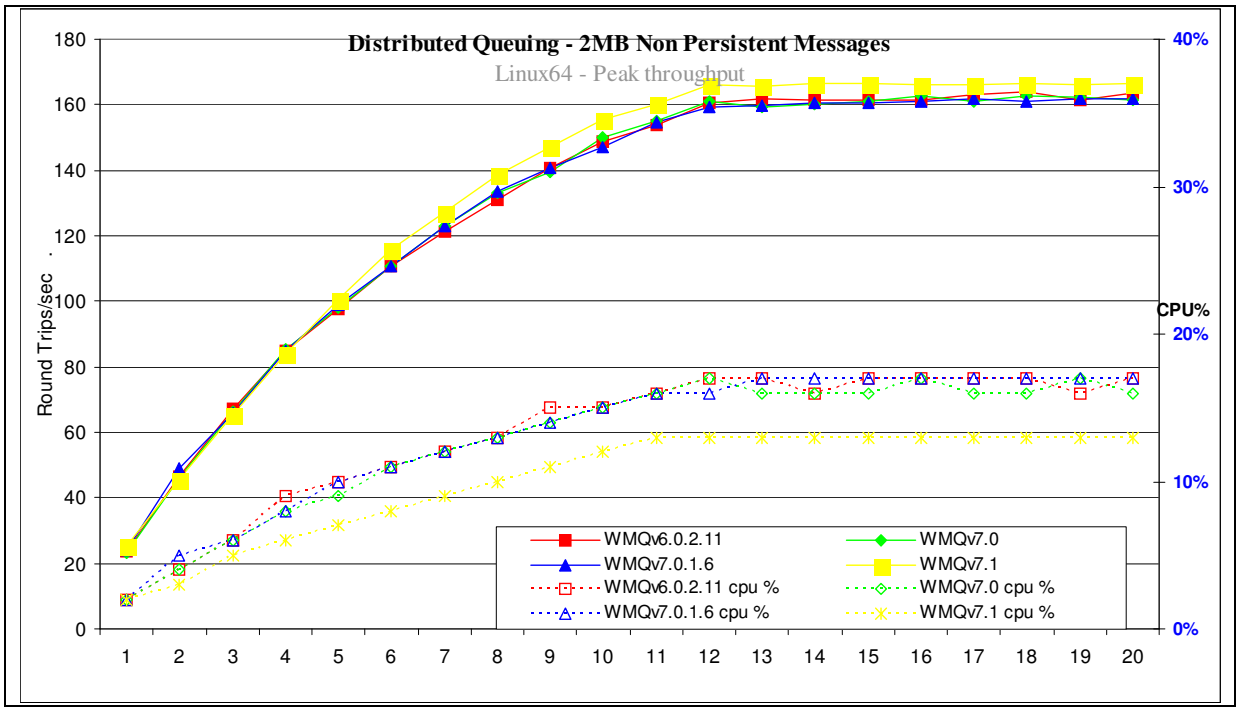


Figure 43 – 2MB non-persistent messages, distributed queuing

Figure 41 and Table 32 shows that the throughput of non-persistent messages is unchanged when comparing version 7.1 to 7.0.1.6 and 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	18	164	0.12	17%
WMQv7.0	16	163	0.1	17%
WMQv7.0.1.6	17	162	0.11	17%
WMQv7.1	18	166	0.12	13%

Table 32 – 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

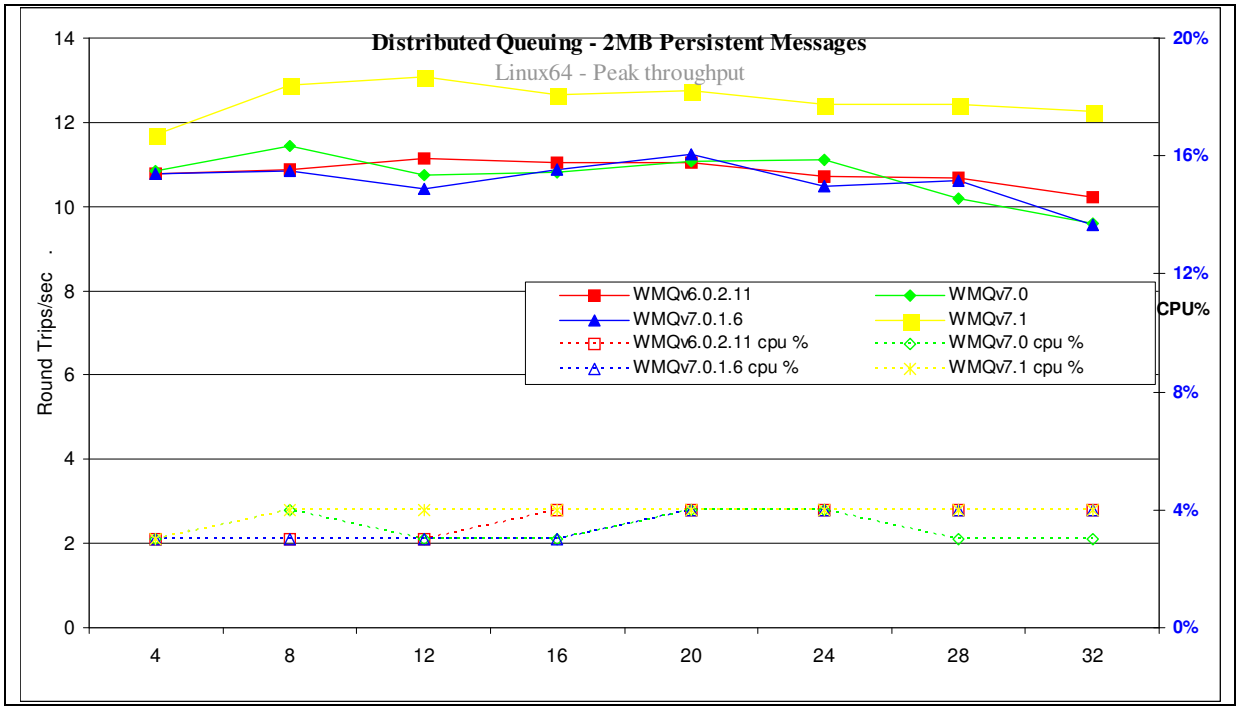


Figure 44 - 2MB persistent messages, distributed queuing

Figure 42 and Table 33 shows that the 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 - 2MB Persistent Messages	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	8	11	0.78	3%
WMQv7.0	8	11	0.75	4%
WMQv7.0.1.6	8	11	0.78	3%
WMQv7.1	8	13	0.66	4%

Table 33 – 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 45 and **Figure 46** 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

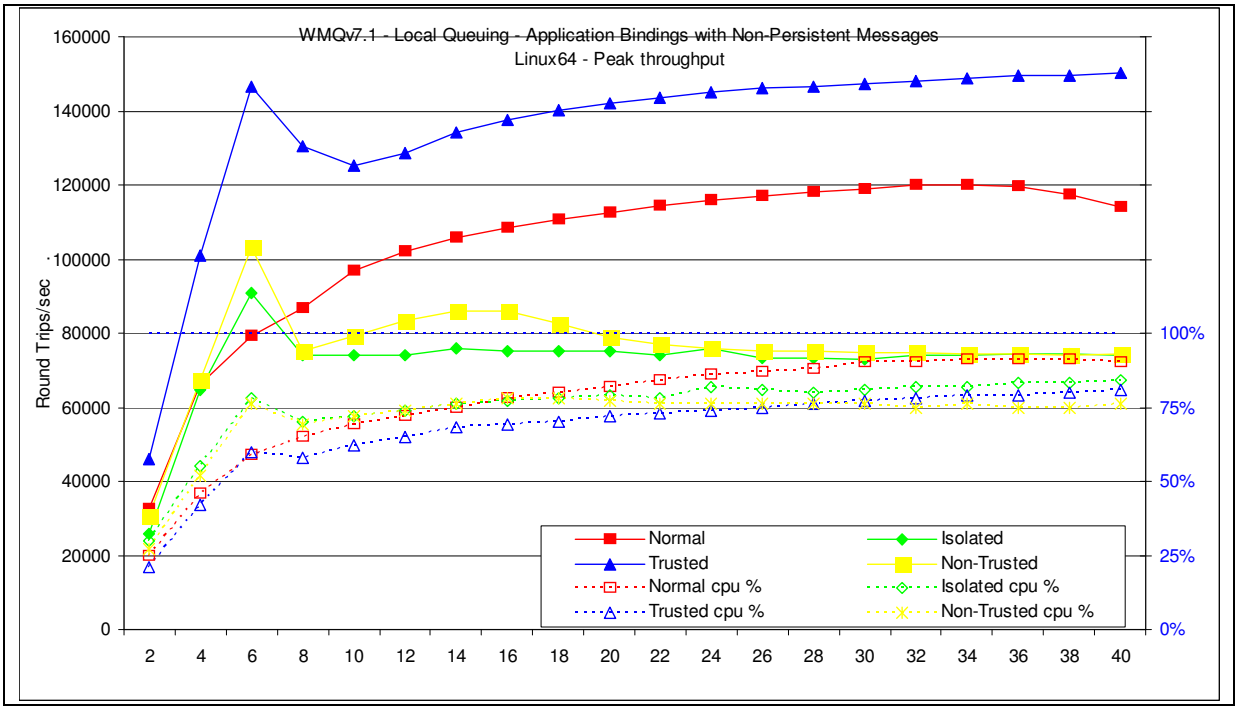


Figure 45 – Application binding, non-persistent messages, local queue manager

Figure 43 and Table 32 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	34	120090	0.0003	91%
Isolated	6	90964	0.00008	78%
Trusted	40	150388	0.00028	81%
Non-Trusted	6	103250	0.00008	76%

Table 34 – Application binding, non-persistent messages, local queue manager

4.1.2 Persistent Messages

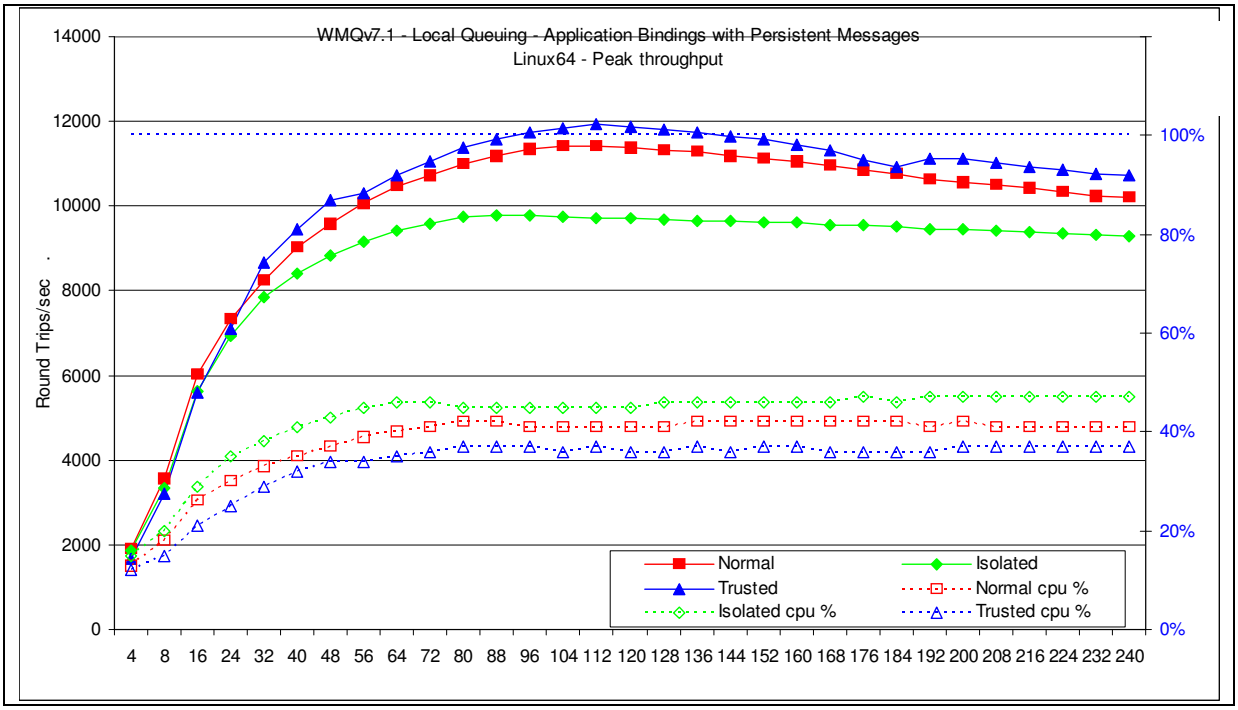


Figure 46 – Application binding, persistent messages, local queue manager

Figure 44 and Table 33 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	104	11418	0.01	41%
Isolated	88	9788	0.011	45%
Trusted	112	11933	0.011	37%

Table 35 – Application binding, persistent messages, local queue manager

.

4.2 Client Channels

Figure 47 and **Figure 48** 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

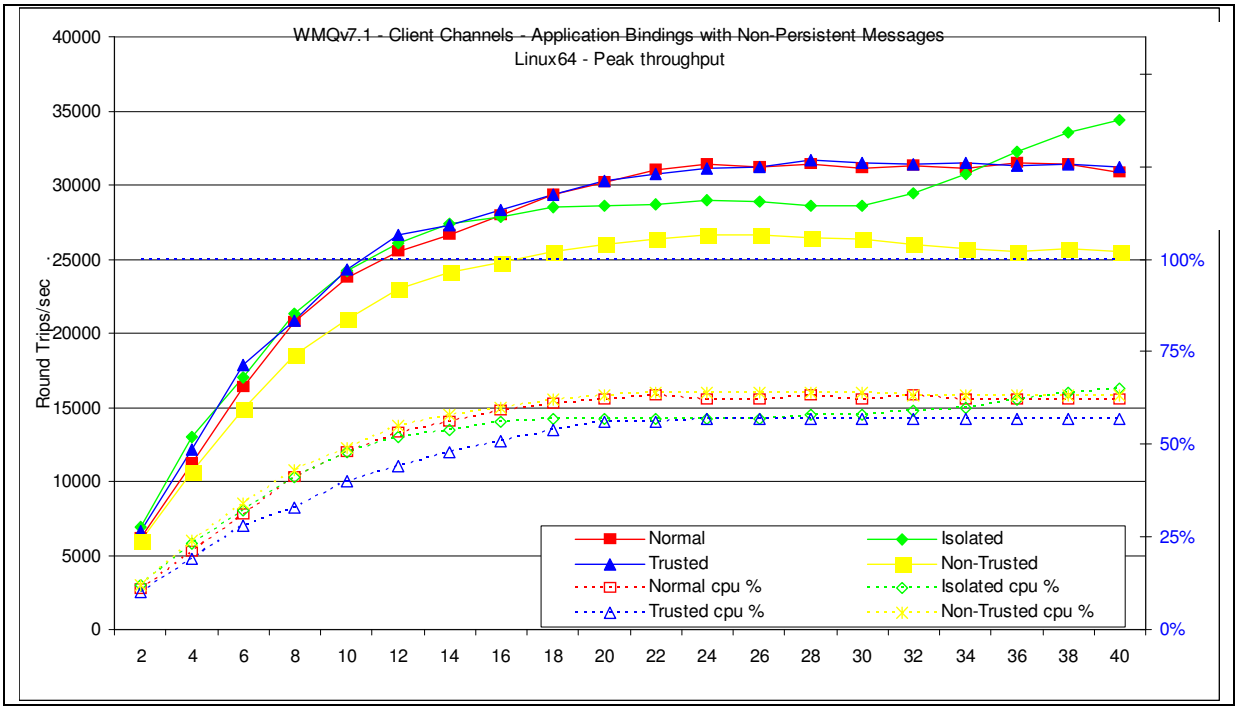


Figure 47 – Application binding, non-persistent messages, client channels

Figure 45 and Table 34 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	36	31502	0.0013	62%
Isolated	40	34365	0.0013	65%
Trusted	28	31654	0.001	57%
Non-Trusted	26	26677	0.0011	64%

Table 36 – Application binding, non-persistent messages, client channels

4.2.2 Persistent Messages

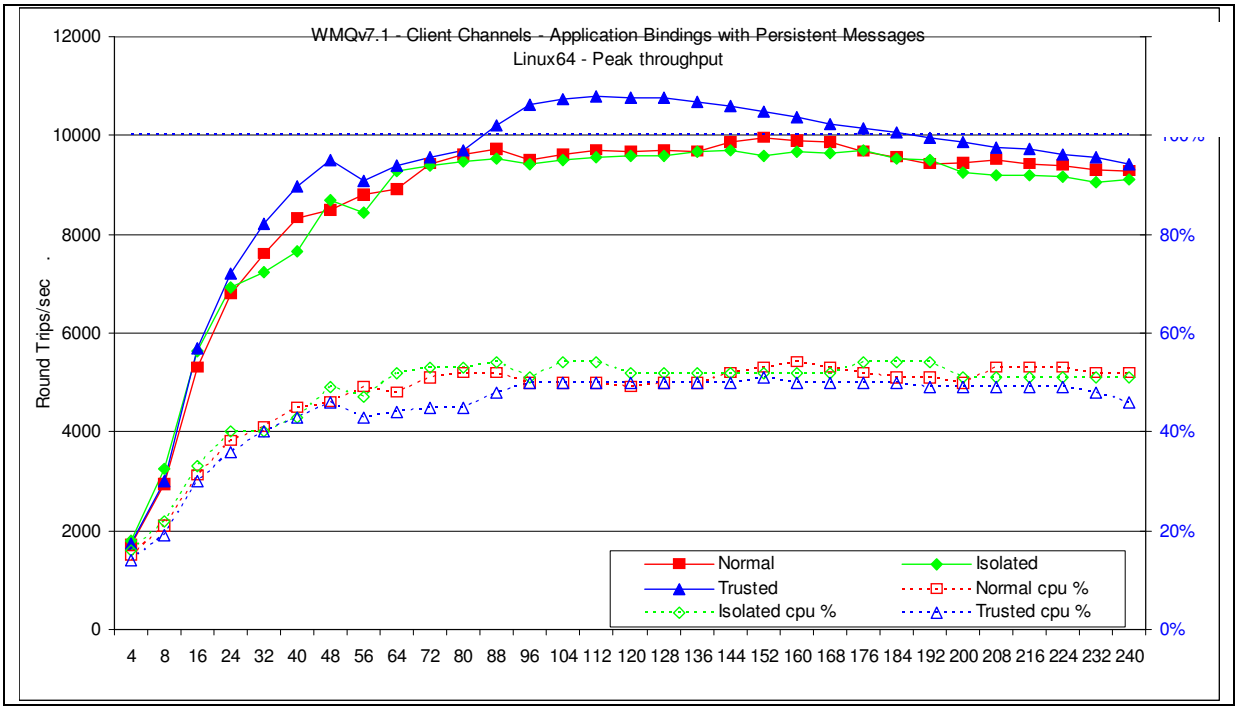


Figure 48 – Application binding, persistent messages, client channels

Figure 46 and Table 35 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	152	9943	0.018	53%
Isolated	176	9708	0.02	54%
Trusted	112	10791	0.012	50%

Table 37 – Application binding, persistent messages, client channels

.

4.3 Distributed Queuing

Figure 48 and **Figure 49** 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

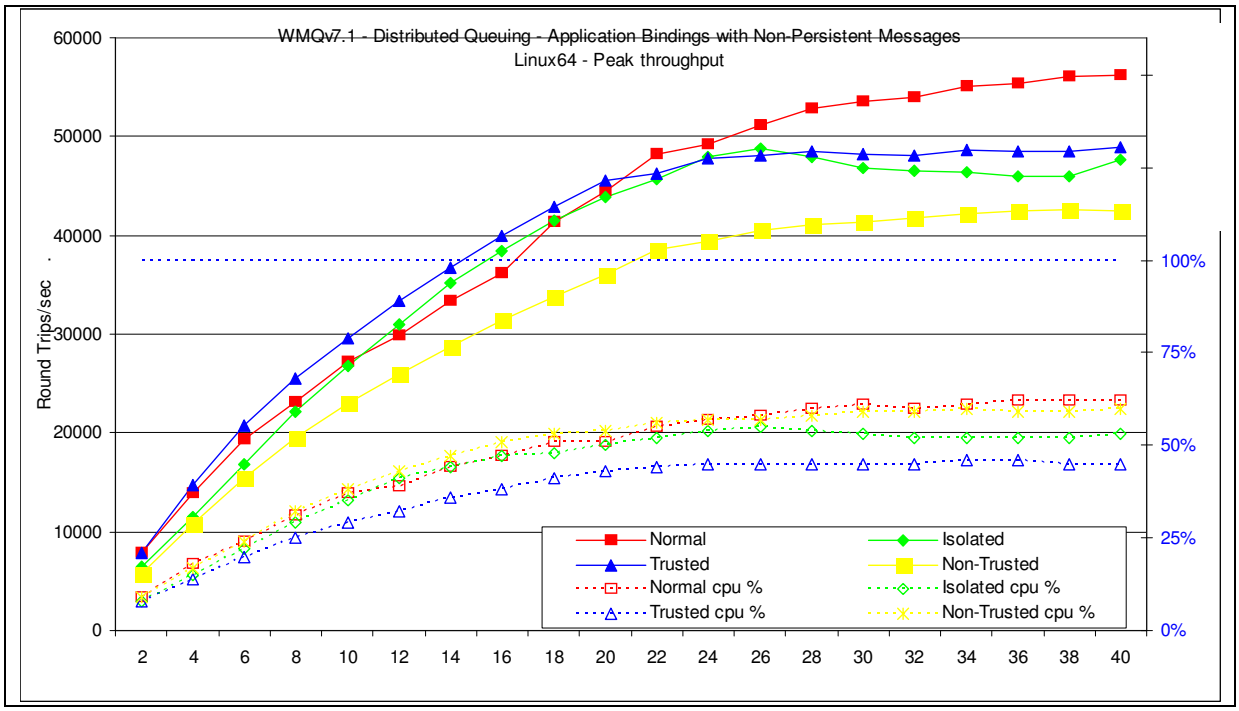


Figure 49 – Application binding, non-persistent messages, distributed queuing

Figure 47 and Table 36 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	40	56211	0.00088	62%
Isolated	26	48757	0.00062	55%
Trusted	40	48871	0.001	45%
Non-Trusted	38	42634	0.0012	59%

Table 38 – 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

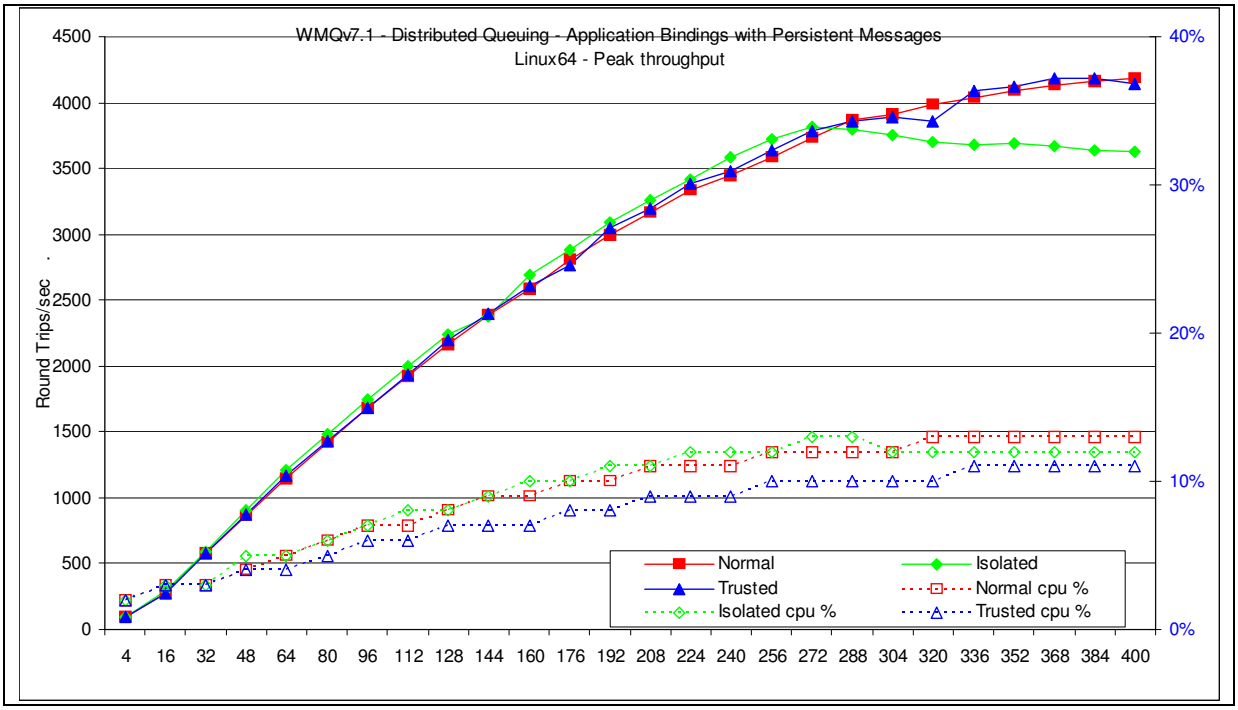


Figure 50 – Application binding, persistent messages, distributed queuing

Figure 48 and Table 37 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	400	4180	0.11	13%
Isolated	272	3814	0.081	13%
Trusted	384	4182	0.1	11%

Table 39 – 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.

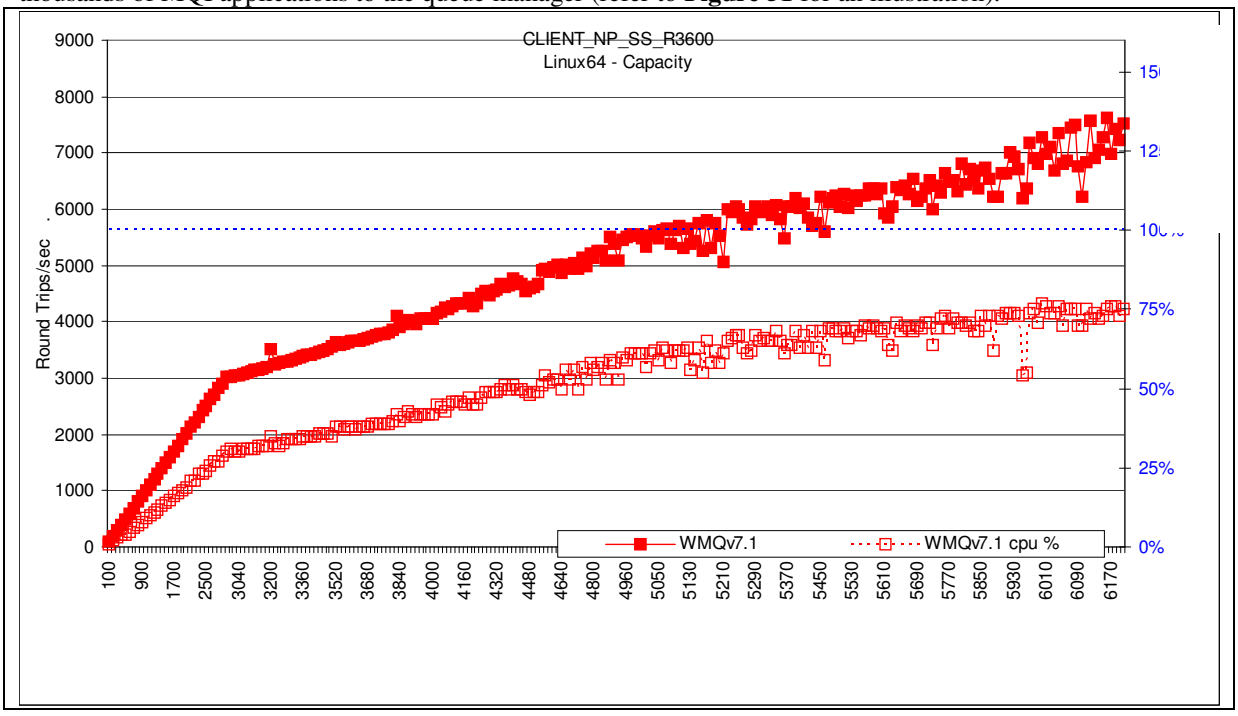
5 Short & Long Sessions

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 51** for an illustration).



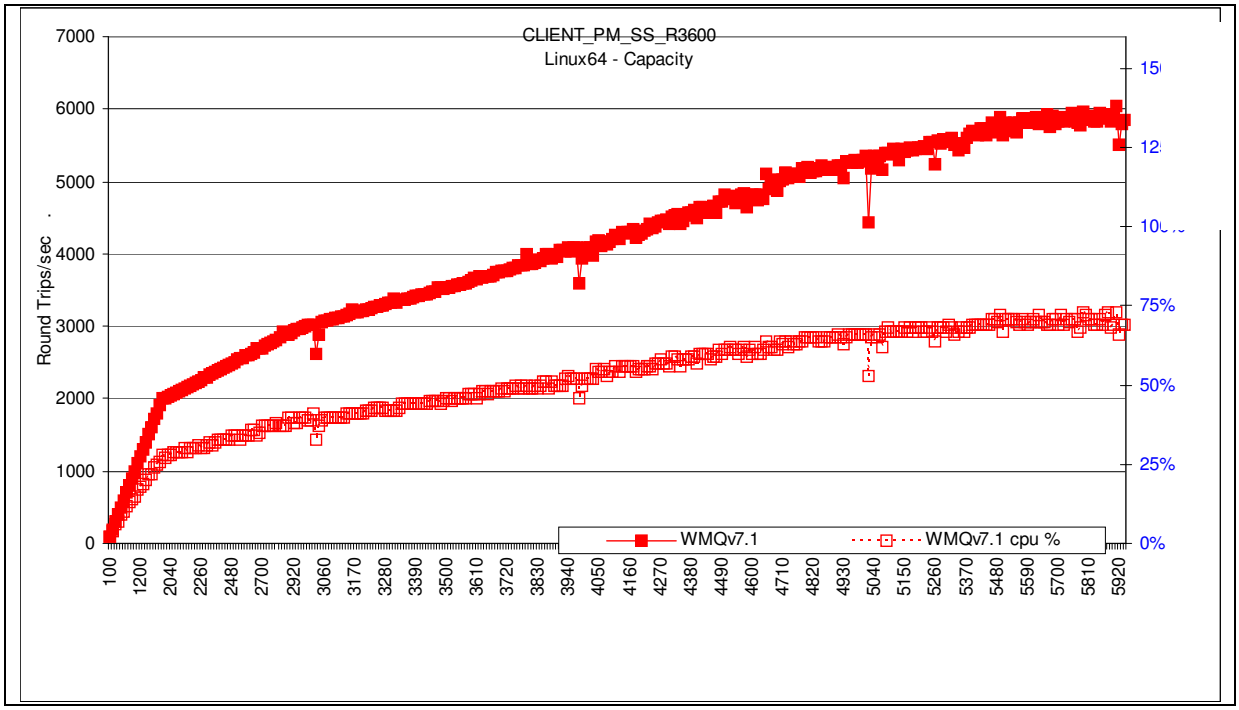


Figure 51 – Short sessions, client channels

Test Name: CLIENT_NP_SS_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv7.1	6070	7443	0.97	75%

Test Name: CLIENT_PM_SS_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv7.1	5750	5941	0.98	71%

Table 40 – 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

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* while additional connections are made. The maximum number of connected applications is likely to be determined by other criteria such as recovery time or manageability. Measurements are also made with smaller number of Client channels where the message insertion rate is increased until the system gets congested. This information is intended to be useful to the reader sizing a system with similar scenarios. These client measurements of V7.1 allocate a separate socket for each client (sharecnv=1 on svrcon channel).

Queue manager configuration for client channels capacity tests:

MaxChannels=50000 (100,000 for clnp_cmax). MQIBINDTYPE=FASTPATH

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
clnp	20	n/a*	32569	0.00072	61%
clnp_r3600	10300	3600	9996	0.00098	32%
clnp_c6000	6000	13500	22388	0.147	59%
clnp max	60000	800	11052	0.0007	61%
cl_persist_c6000	6000	4000	6541	0.582	44%
clnp_cmax_no_correllid	31000	400 500	3523 4342	0.0018 0.1059	43% 70%
cl_persist_cmax_no_correllid	31000	200	1596	0.01	32%

Table 41 – Capacity measurements, client channels

* *There was no delay between the response to the previous message and the insertion of the next message with 38 clients.*

The maximum message throughput is achieved when there are a small number of requester applications. The clnp_3600 measurement peaks when the queue of input messages waiting to be processed by the Server application builds up because the server application threads can no longer keep up with the demand. Although this ensures the server threads are always busy, the messages are being spilt from the Queue buffer to the file system and possibly to the disk. Each client uses a thread in the AMQRRMPA processes and the management of lots of threads and lots memory objects results in a larger CPU cost to handle each message.

Measurements normally use a Get by Correlation_Id from a common reply queue for all clients whereas the tests labelled 'no_correlid' have a separate reply queue per client. Each additional Client needs a thread in the AMQRMPPA process. Using a separate queue per client needs additional shared memory per client.

6.1.1 Client Channels – Memory

The clnp-cmax test was run a machine where memory could be stolen by a page-fix program. Each Client inserts one 2K byte message a minute and the number of clients increases until the average response time exceeds a second. The table records the maximum number of clients that the Queue manager could process messages with a response time of under a second

memory	V6.0.2.11	V7.0.1.6	V7.1
1GB	9000	4500	4500
2GB	22500	11500	11500
3GB		18500	18500
4GB	45000	25000	25000
6GB	>64000	38000	38500

Linear approximations of these points enable the amount of storage per client and the cost of the first client (including operating system and Queue manger) to be calculated

	V6.0.2.11	V7.0.1.6	V7.1
Op_Sys + QM + first client	189MB	284MB	310MB
Additional clients	84KB	149KB	147KB

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* while applications are being attached. For the same number of server channel pairs, a faster message rate gives a higher total message throughput over each channel pair. This information is intended to be useful to the reader sizing a system with similar scenarios.

Queue manager and log configuration for distributed queuing capacity tests:

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

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

Test name:	Apps	Rate/app/hr	Round Trips/sec	Response time (s)	CPU
dqnp	20	n/a*	47414	0.00051	48%
dqnp_r3600	16750	3600	15995	0.00055	20%
dqnp_q1000	1000	60000	16486	0.00066	30%
dqnp_qmax	10000	12000	24935	0.00618	82%
dq-persist-qmax	10000	3360	1148	0.0628	31%
dq-persist_q1000	1000	13280	3670	0.237	23%

Table 42 – 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 dqnp and dqnp_r3600 both used a total of 4 pairs of Sender/Receiver pairs of channels between queue managers while the dqnp_qmax and dq_persist_q4000 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.

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 Linux 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 x3650:	Server system (Device under test)
Model:	x3650 M3 8864 4RG
Processor:	2.8GHz Intel Xeon x5660
Architecture:	2 x 6 core CPU
Memory (RAM):	32GB
Disk:	2 SAN disks on DS8700 (5GB each, 1 queue, 1 log)
Network:	10Gbit Ethernet Adapter

IBM x3850:	Driver system
Model:	x3850 M2 8864 4RG
Processor:	2.93GHz Intel Xeon x7350
Architecture:	4 x quad core CPU
Memory (RAM):	32GB
Disk:	2 SAN disks on DS8700 (5GB each, 1 queue, 1 log)
Network:	10Gbit Ethernet Adapter

The machines under test are connected to a SAN via a dedicated SVC. The SVC provides a transparent buffer between the server and SAN that will smooth any fluctuations in the response of the SAN due to external workloads. The server machines are connected via a fibre channel trunk to a 8Gb Brocade DCX director. The speed of each server is dictated by the server's HBA (typically 2Gb). 5GB generic LUNs are provisioned via SVC. The SVC is a 2145-8G4 which connects to the DCX at 4Gb. The SAN storage is provided by an IBM DS8700 which is connected to the DCX at 4Gb.

8.3 Software

Linux 64 bit:	Red Hat Enterprise Linux AS release 5.5 (Tikanga)
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

10 Multicast

A number of Publish/Subscribe scenarios were run to compare the performance of MQ v7.1 using MQ Pub/Sub and Multicast. The MQ C client was used to drive the tests. All the scenarios use Client bindings.

10.1 Single Publisher, Single Subscriber

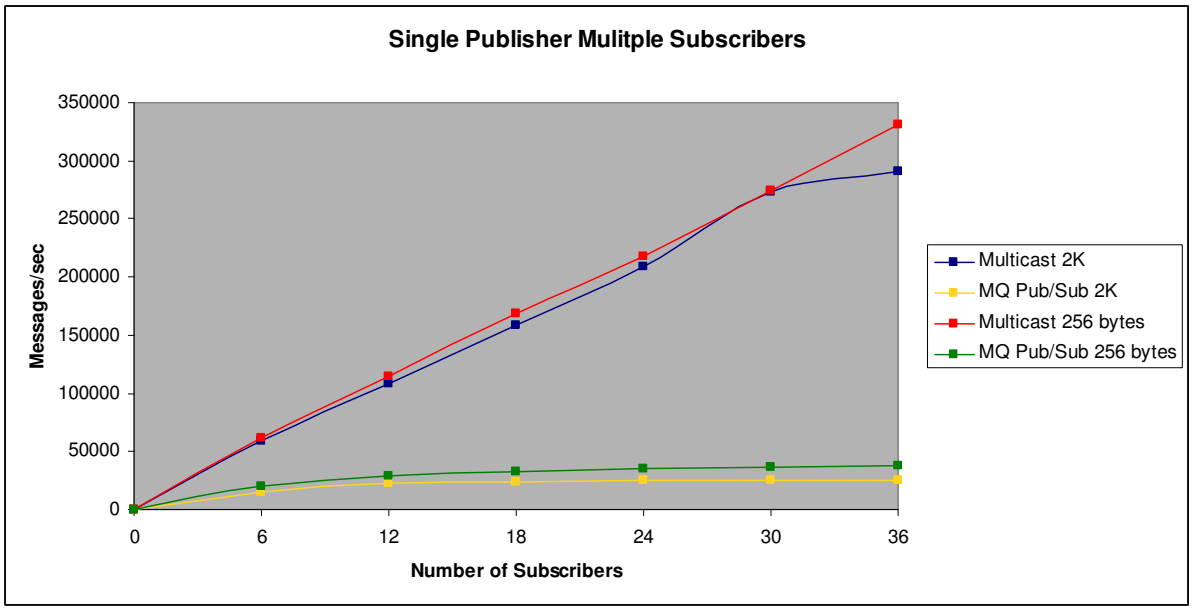
The Publisher and Subscriber were run on separate client machines. The test measures the maximum publication rate that can be achieved. The message size used for this test was 2KB.

	Msgs/sec	CPU QManager	CPU Publisher	CPU Subscriber
MQ Pub/Sub	3020	9.2%	5%	4%
Multicast	8550	0.2%	26%	12%

The limiting factor in the MQ Pub/Sub case is the speed at which the MQ QManager can publish the message to the registered subscriber together with the latency between Publisher and QManager.
The limiting factor in the Multicast case is the Publisher CPU. The multicast Publisher is single threaded and can only use one CPU of the four available on the client machine, hence when the CPU reaches 25% the test is CPU limited.

10.2 Single Publisher, Multiple Subscribers

This scenario measures the effect of adding Subscribers when there is a single Publisher.



The message rate shown is the number of messages per second published by the Publisher plus the number of messages per second received by all the subscribers. For example, if the publisher were publishing at 100 msgs/sec to 6 subscribers the throughput shown would be 700 msgs/sec.
The graph shows that adding subscribers has very little effect in the multicast case, but for MQ Pub/Sub the publication rate drops as subscribers are added.

The tables below show the effect of adding subscribers on publication rate.

Number of Subscribers	MQ Pub/Sub Publications/sec	Multicast Publications/sec	Improvement
6	2807	8810	3.1x
12	2222	8848	4.0x
18	1731	8860	5.1x
24	1418	8737	6.2x
30	1194	8872	7.4x
36	1023	8939	8.7x

Table 43 - Single Publisher Multiple Subscribers 256 byte message

Number of Subscribers	MQ Pub/Sub Publications/sec	Multicast Publications/sec	Improvement
6	2197	8367	3.8x
12	1701	8361	4.9x
18	1272	8360	6.6x
24	1006	8370	8.3x
30	827	8816	10.7x
36	687	7863	11.5x

Table 44 - Single Publisher Multiple Subscribers 2048 byte = 2K message

10.3 Machine configuration

MQ QManager machine
An xSeries 350 4 x 2.8GHz Intel Xeon CPUs with 8GB of RAM.
Linux Redhat 3.4.6

Publisher machine
An xSeries 3850 4 x 3169 MHz Intel Xeon CPUs with 4GB of RAM.
Linux Redhat 4.1.2

Subscriber machines
Subscribers were hosted on up to 6 driver machines of varying powers.

All machines were connected over a 1Gb Ethernet LAN which was sufficient to handle the data rates without introducing a bottleneck.