WebSphere MQ AIX v7.1
Performance Evaluations

Version 1.2

February 2012

Fred Preston, Craig Stirling , Peter Toghill.

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

Property of IBM

Please take Note!

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

First Edition, December 2011.

This edition applies to *WebSphere MQ for Windows 64bit v7.1* (and to all subsequent releases and modifications until otherwise indicated in new editions).

© Copyright International Business Machines Corporation 2011. All rights reserved.

Note to U.S. Government Users
Documentation related to restricted rights.
Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

Notices

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

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

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

WARRANTY AND LIABILITY EXCLUSION
The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

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

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

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

wanting to understand the performance characteristics of *WebSphere MQ for AIX 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 AIX.
- Want to understand the performance limits of WebSphere MQ for AIX v7.1.
- Want to understand what actions may be taken to tune WebSphere MQ for AIX bit.

The reader should have a general awareness of the Windows 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 Power 6 P570 8core cpu LPAR with 16GB of RAM was used as the Device under test.

A Power 7 p750 8 core cpu LPAR with 32GB of RAM was used as the Driver

How this document is arranged

Pages: 1-18

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: 25-46

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: 47-52

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

Tuning Recommendations

Pages: 57-

Measurement Environment

Pages: 62 64

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: 64

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

CONTENTS

TABLES

FIGURES

1 Overview

WebSphere MQ v7.1 on AIX has improved performance especially for smaller messages using a small number of queues. For 2KB messages, almost every test shows improvements over earlier versions of WMQ.

Comparing the Message Rate Graphs for 2K byte Non-Persistent messages used in Local, Client, and Distributed Queuing environments, Version 7.1 has 28% higher throughput than V6.0.2.11, 30% higher throughput than V7.0, and 35% higher than V7.0.1.6

Comparing the Message Rate Graphs for 2K byte Persistent messages used in Local, Client, and Distributed Queuing environments, Version 7.1 has 64% higher throughput than V6.0.2.11, 36% higher throughput than V7.0, and 48% higher throughput than V7.0.1.6

2 Performance Headlines

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

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

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

All tests stop automatically after the response time exceeds 1 second.

2.1 Local Queue Manager Test Scenario

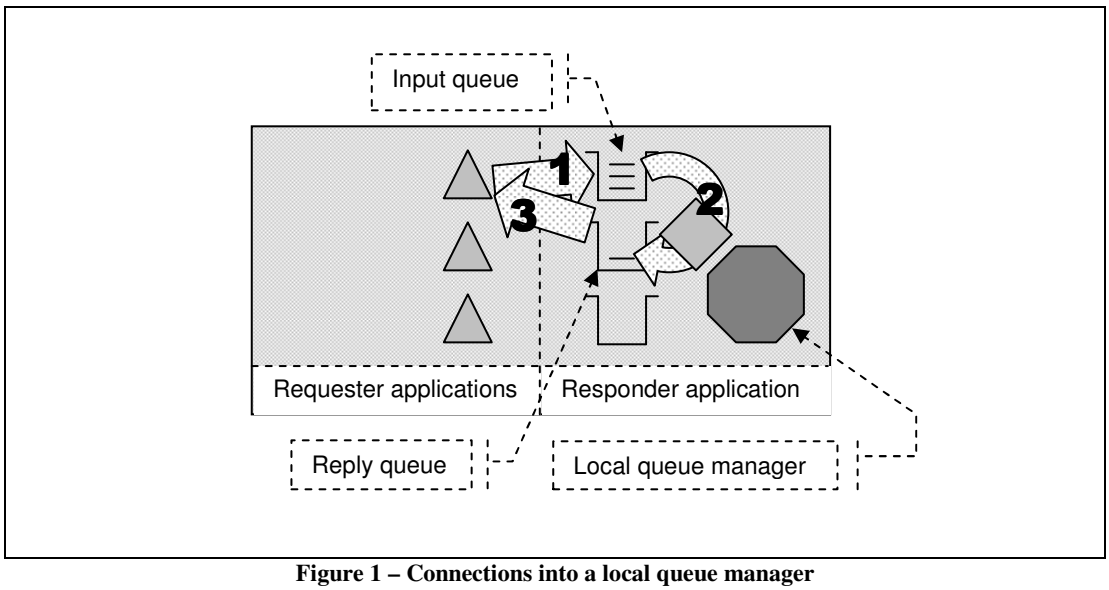


Figure 1 – Connections into a local queue manager

1) The Requester application puts a message to the common input queue on the local queue manager, and holds on to the message identifier returned in the message descriptor. The Requester application then waits indefinitely for a reply to arrive on the common reply queue.

2) The Responder application gets messages from the common input queue and places a reply to the common reply queue. The queue manager copies over the message identifier from the request message to the correlation identifier of the reply message.

3) The Requester application gets a reply from the common reply queue using the message identifier held from when the request message was put to the common input queue, as the correlation identifier in the message descriptor.

Non-persistent and persistent messages were used in the local queue manager tests, with a message size of 2KB. The effect of message throughput with larger messages sizes is investigated in the "***Large Messages***" section.

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

2.1.1 Non-persistent Messages – Local Queue Manager

Figure 2, Figure 3 and Figure 4 shows the non-persistent, non-persistent non-trusted and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario (see Figure 1 on the previous page) for different production levels of WebSphere MQ (versions 7.1, 7.0.1.6, 7.0 and 6.0.2.11).

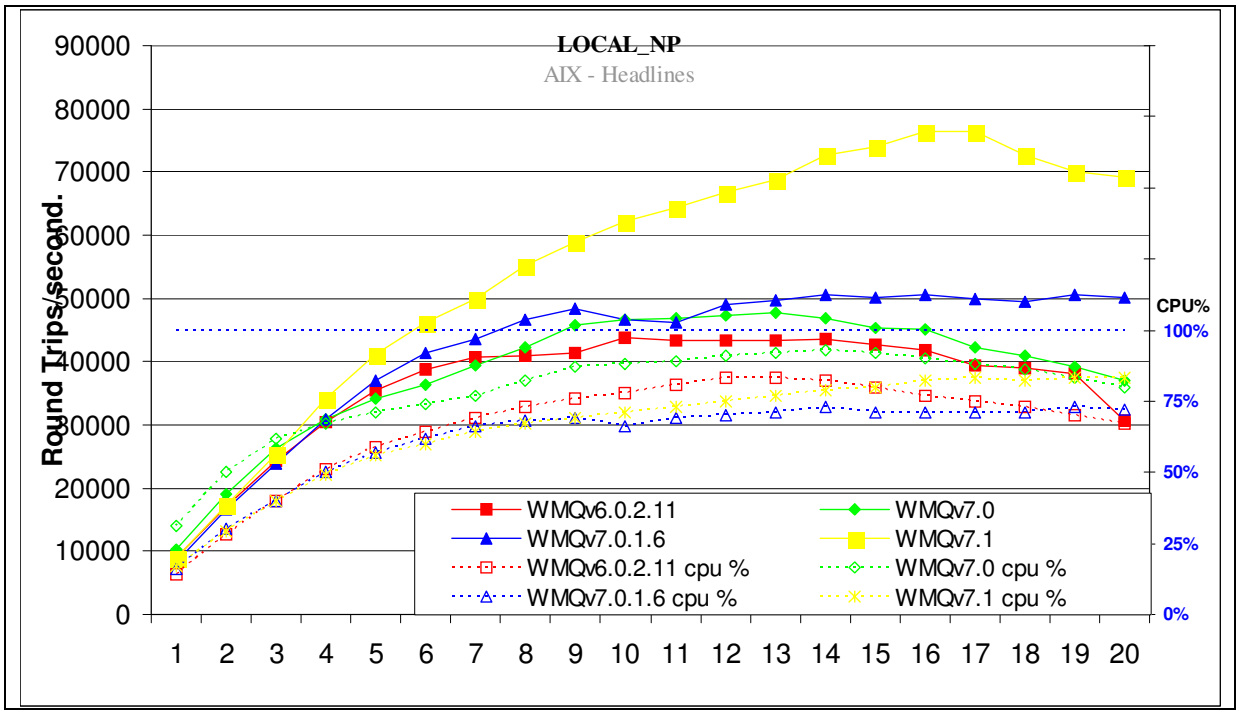


Figure 2 – Performance headline, non-persistent messages and local queue manager.

Figure 2 and Table 1 show that the throughput of non-persistent messages has increased by 32% when comparing version 7.1 to V7.0

Test Name: LOCAL_NP	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	10	43767	0.00026	78%
WMQv7.0	13	47839	0.00035	92%
WMQv7.0.1.6	14	50674	0.00026	73%
WMQv7.1	17	76457	0.00027	83%

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

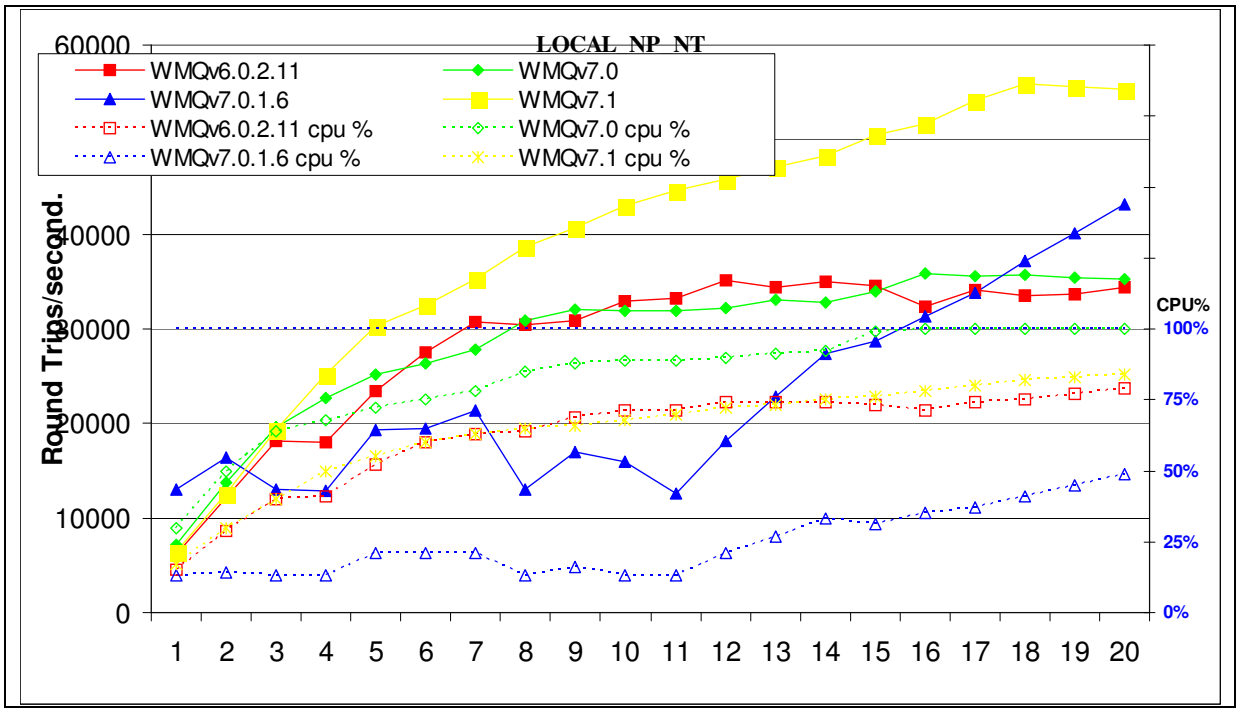


Figure 3 Performance headline, non-persistent, non-trusted messages and local queue manager.

Figure 3 and Table 2 shows that the throughput of non-persistent, non-trusted messages (shared bindings - MQIBINDTYPE=NORMAL) has increased by 35% when comparing version 7.1 to v7.0.

Test Name: LOCAL_NP_NT	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	12	35105	0.0003	74%
WMQv7.0	16	35900	0.00052	100%
WMQv7.0.1.6	20	43147	0.00024	49%
WMQv7.1	18	55900	0.00037	82%

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

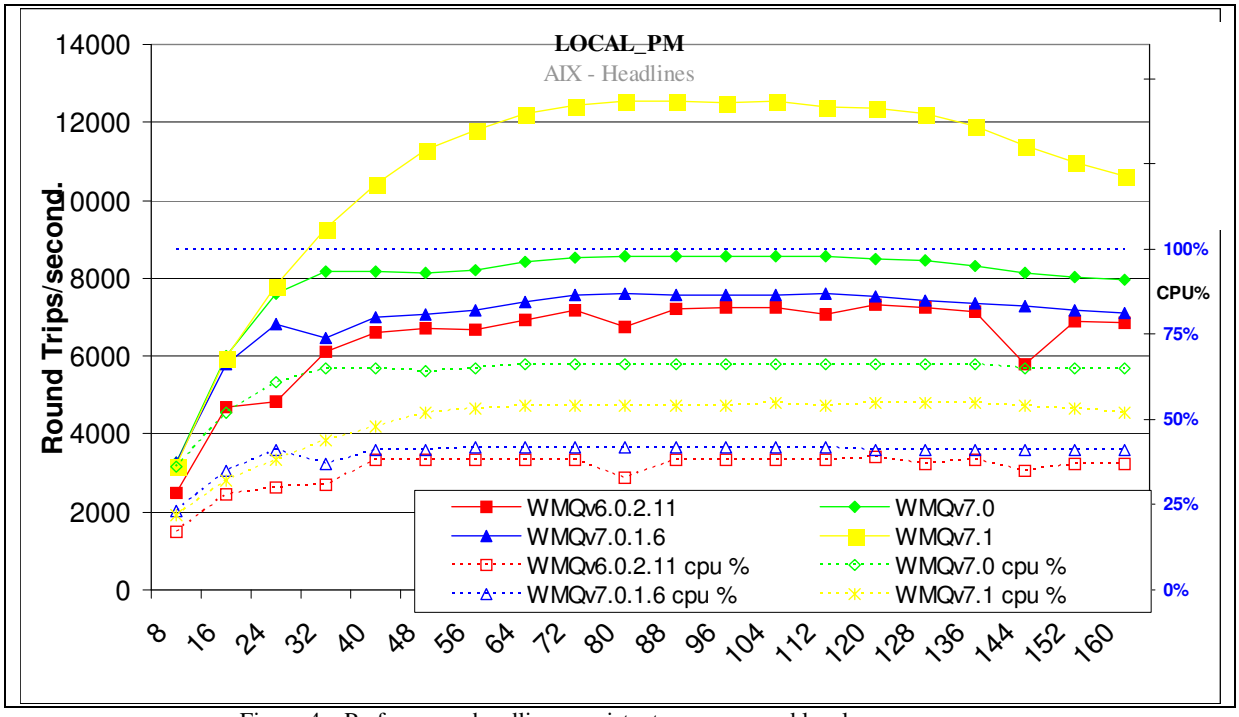


Figure 4 – Performance headline, persistent messages and local queue manager

Figure 4 and Table 3 shows that the throughput of persistent messages has increased by 30% when comparing version 7.1 to V7.0.

Test Name: LOCAL_PM	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	120	7305	0.019	39%
WMQv7.0	80	8571	0.011	66%
WMQv7.0.1.6	112	7605	0.017	42%
WMQv7.1	80	12554	0.0076	54%

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 Scaling – Local Queue Manager

These measurements used an LPAR on the Power 7 machine (used elsewhere in this report as the Driver system) as the Driver/Queue manager. This application model of using a single server input Queue stresses the locking characteristics of the system and MQ V7.1 can utilise more cores than V7.0.1.6 These charts show an improvement in scalability provided by Version 7.1

2.1.4.1 Scaling – Local non Persistent, non Trusted

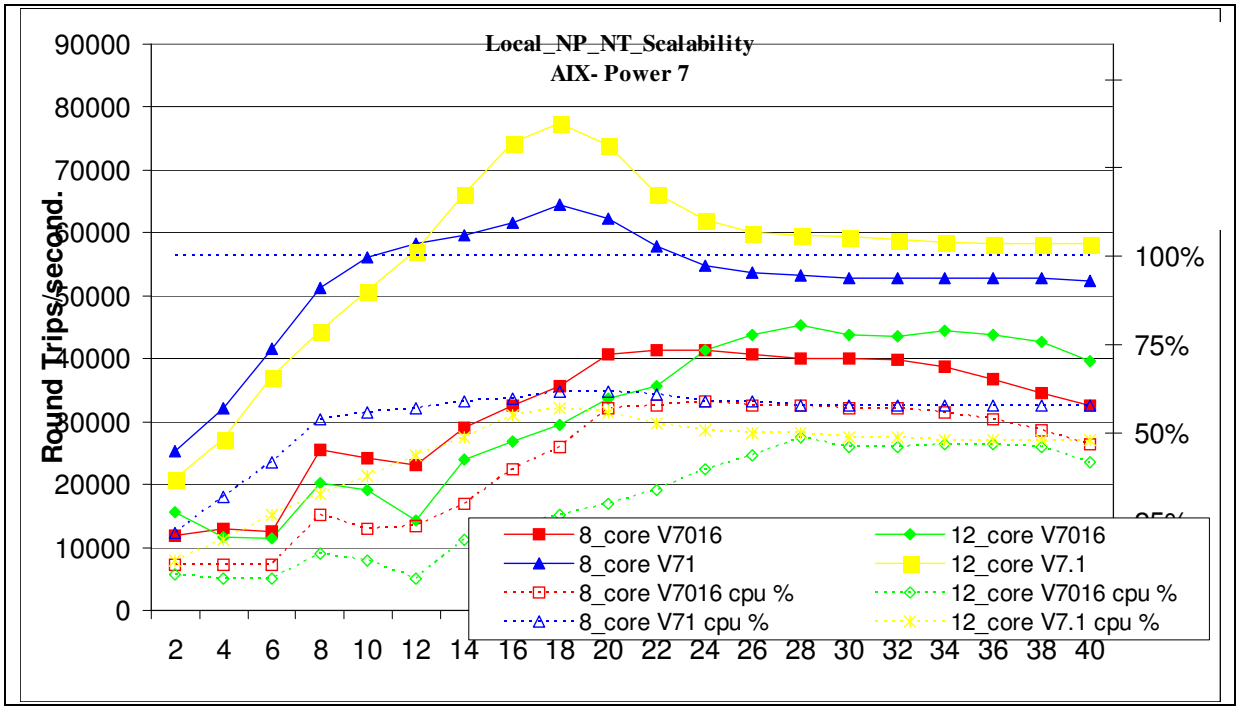


Figure 5 Scaling Local NP NT

Test Name: Local_NP_NT_Scalability	Apps	Round Trips/Sec	Response time (s)	CPU
12_core V7.1	18	77364	0.0003	57%
8_core V7016	24	41452	0.00035	59%
12_core V7016	28	45222	0.00034	49%
8_core V71	18	64426	0.00035	62%

Table 4 Scaling Local NP Non trusted

Figure 5 and Table 4 shows V7.0.1.6 increased peak throughput by 9% when increasing the number of cores from 8 to 12. Version 7.1 gets a 20% increase in peak throughput when changing from 8 to 12 cores. Version 7.1 achieves a higher CPU utilisation across this measurement range.

2.1.4.2 Scaling – Local non Persistent V7.1

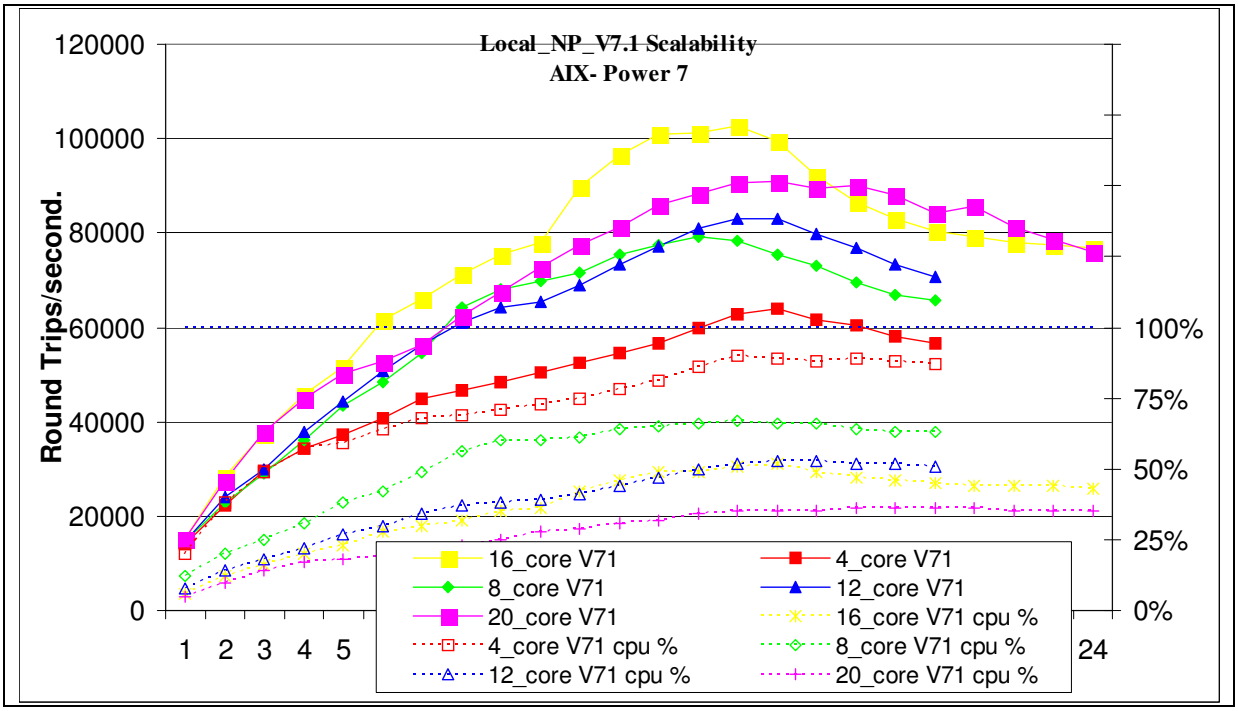


Figure 6 - Scaling Local NP V7.1

Test Name: Local_NP_V7.1 Scalability	Apps	Round Trips/Sec	Response time (s)	CPU
16_core V71	15	102730	0.00015	51%
4_core V71	16	63859	0.00032	89%
8_core V71	14	79123	0.0002	66%
12_core V71	15	83174	0.0002	52%
20_core V71	16	90847	0.00018	35%

Table 5 - Scaling NP V7.1 -

Figure 6 and Table 5 shows that Version 7.1 throughput increases as the number of cores is increased up to 16 but throughput degrades with 20 cores. Peak throughput with 8 cores is 24% higher than 4 cores. 12 cores is 30% higher than 4 cores. 16 cores is 60% higher than 4 cores. 20 cores is 42% higher than 4 cores.

2.1.4.3 Scaling – Local non Persistent V7.0.1.6

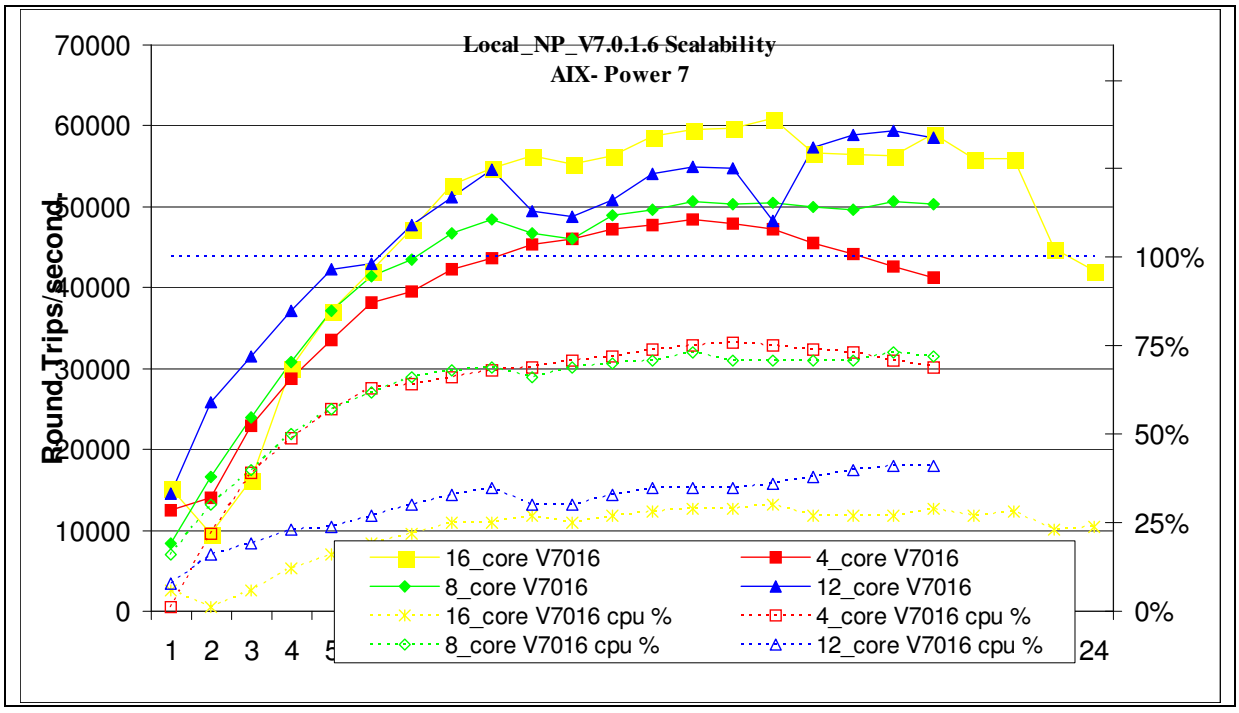


Figure 7 - Scaling Local NP V7.0.1.6

Test Name: Local_NP_V7.0.1.6 Scalability	Apps	Round Trips/Sec	Response time (s)	CPU
16_core V7016	16	60910	0.00016	30%
4_core V7016	14	48463	0.00032	75%
8_core V7016	14	50674	0.00026	73%
12_core V7016	19	59455	0.00023	41%

Table 6 - Scaling Local NP V7.0.1.6

Figure 7 and Table 6 show Version 7.0.1.6 throughput increases with the number of cores up to 12 but degrades with 16 cores. The maximum throughput with 8 cores is 5% higher than 4 cores. 12 cores is 22% higher than 4 cores. 16 cores is 25% higher than 4 cores but the intermediate points mean the overall curve is not better.

These Local scaling charts have shown that V7.1 scales across more cores than V7.0.1.6
This scaling is limited by the message rate per second that can be processed on a single queue. Other application models that have multiple queues will be able to exploit more than 20 cores per queue manager.

2.2 Client Channels Test Scenario

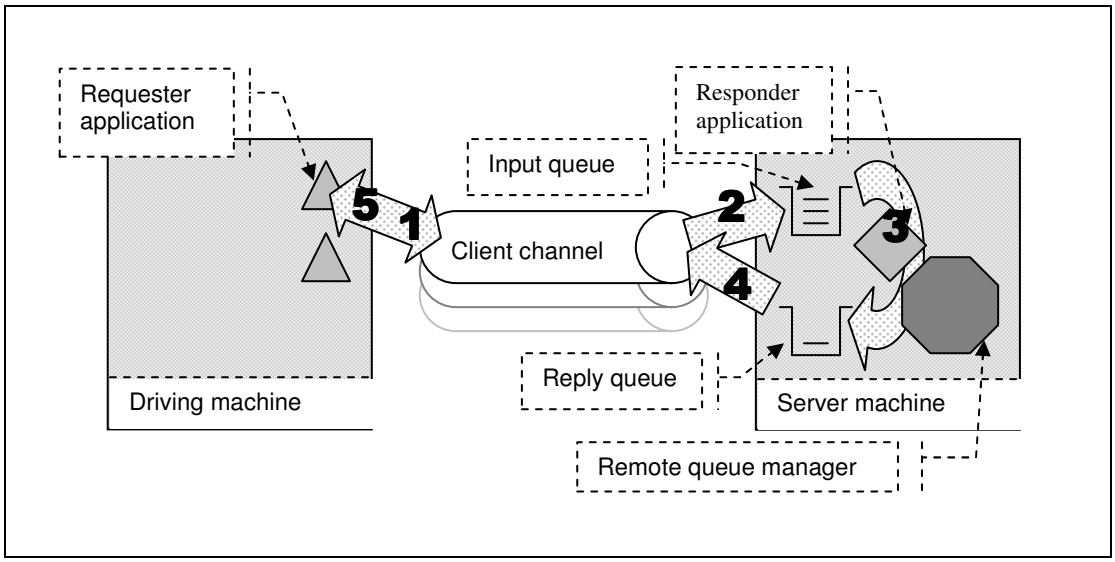


Figure 8 – MQI-client channels into a remote queue manager

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 9, Figure 10 and Figure 11 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 8 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).

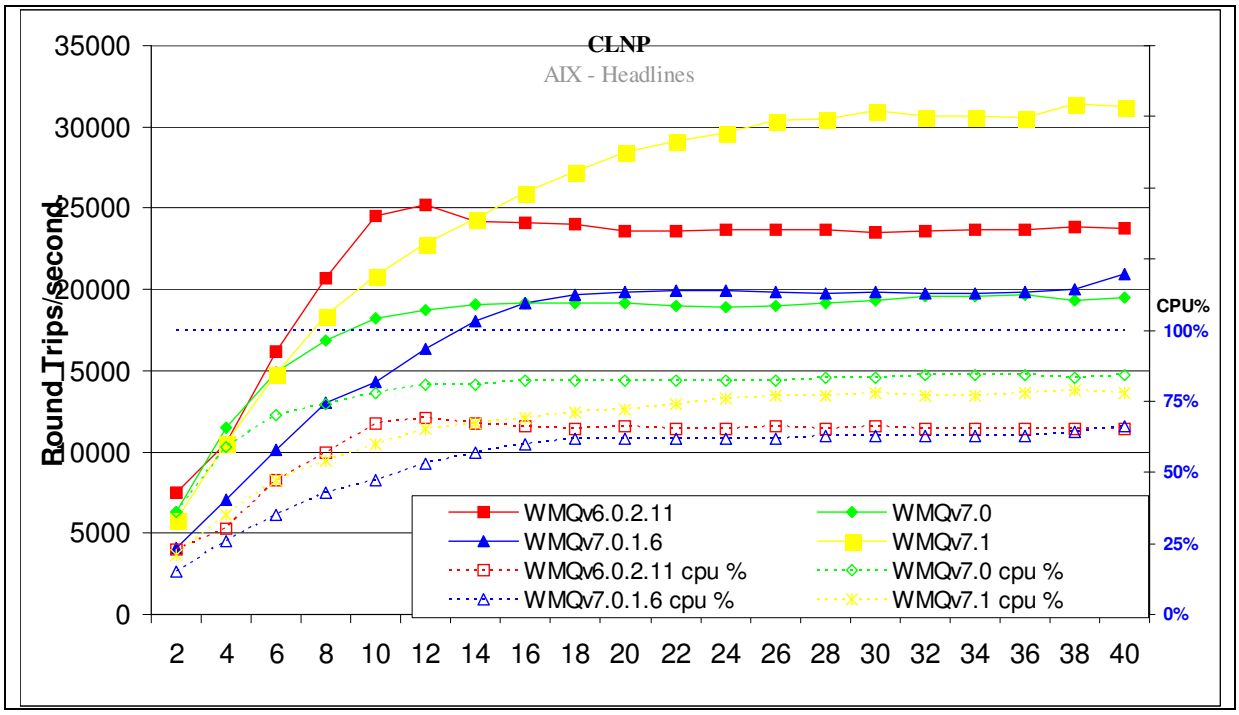


Figure 9 – Performance headline, non-persistent messages and client channels

Figure 9 and Table 7 show that the throughput of non-persistent messages has increased by 40% when comparing version 7.1 to V7.0 and by 14% when comparing version 7.1 to 6.0.2.11.

Test Name: CLNP	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	12	25244	0.00051	69%
WMQv7.0	36	19663	0.0021	84%
WMQv7.0.1.6	40	20958	0.0021	66%
WMQv7.1	38	31383	0.0015	79%

Table 7 – 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

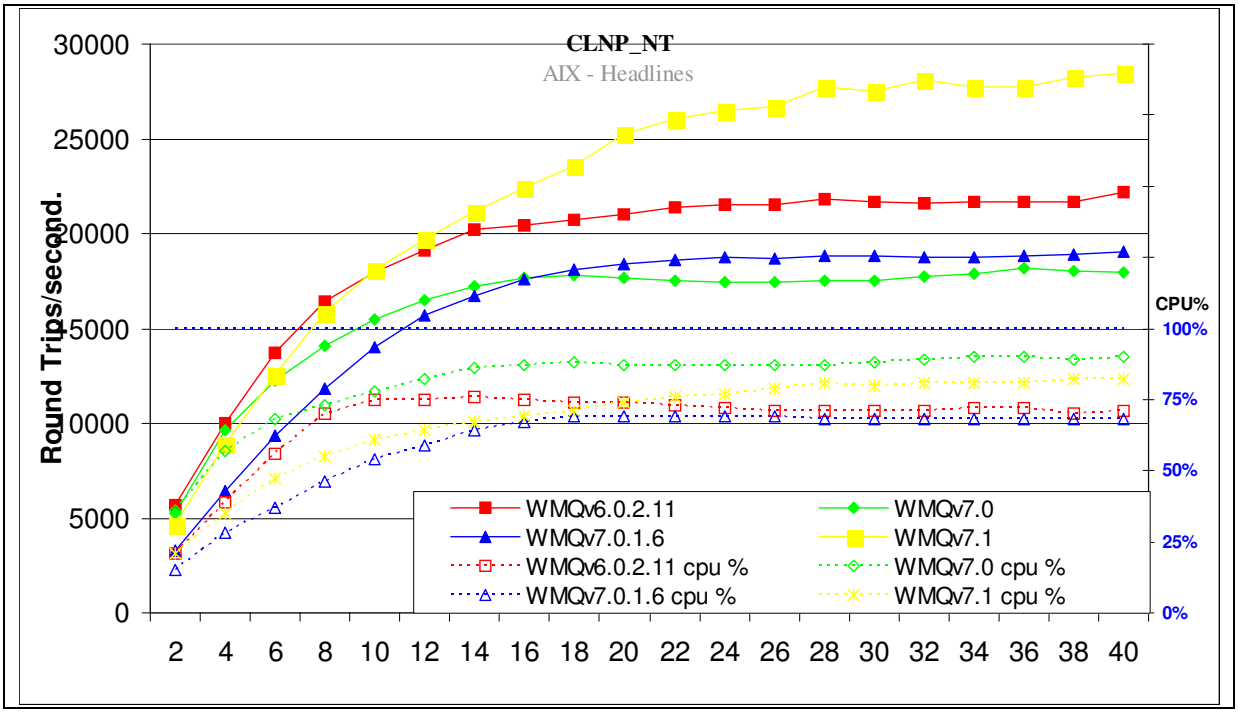


Figure 10 – Performance headline, non-persistent messages with non-trusted client channels

Figure 10 and Table 8 shows that the throughput of non-persistent, non-trusted messages (shared bindings - MQIBINDTYPE=NORMAL) has increased by 39% when comparing version 7.1 to V7.0 and improved by 17% when comparing version 7.1 to 6.0.2.11.

Test Name: CLNP_NT	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	40	22212	0.0021	71%
WMQv7.0	36	18179	0.0023	90%
WMQv7.0.1.6	40	19086	0.0023	68%
WMQv7.1	40	28494	0.0016	82%

Table 8 – 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

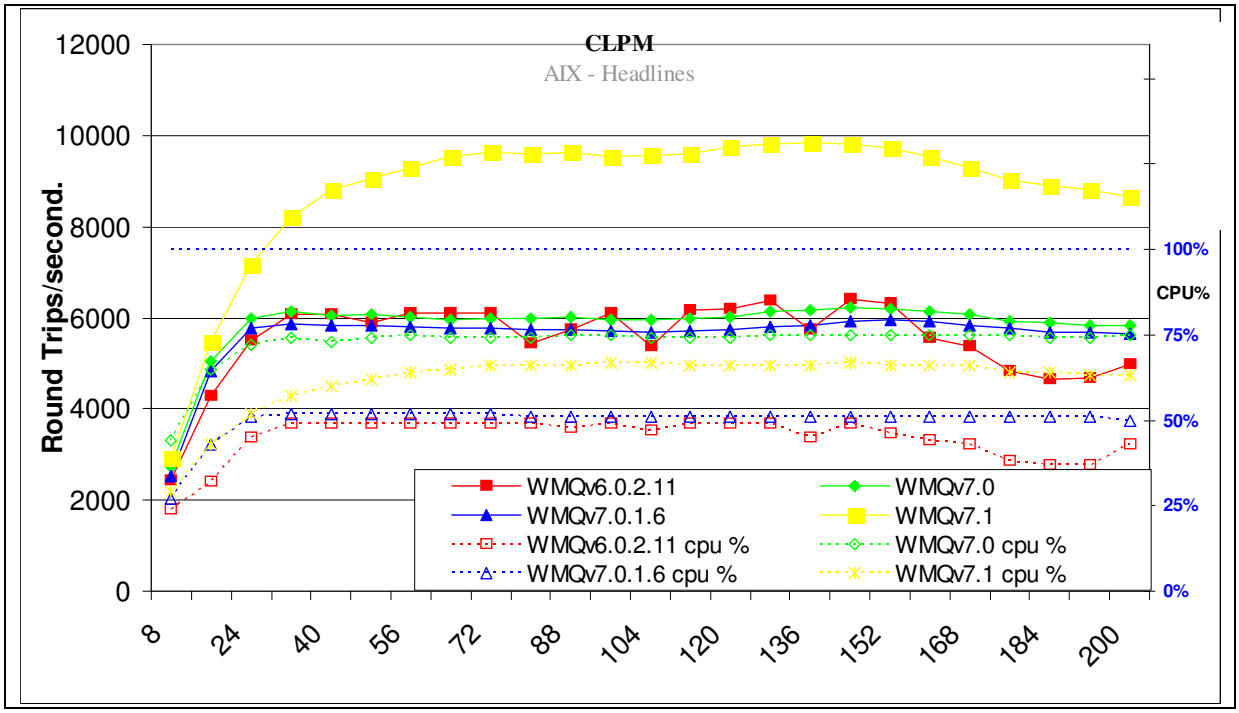


Figure 11 – Performance headline, persistent messages and client channels

Figure 11 and Table 9 shows that the throughput of persistent messages has increased by over 45% when comparing version 7.1 to V7.0 or V6.0.2.11

Test Name: CLPM	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	144	6408	0.026	49%
WMQv7.0	144	6223	0.027	75%
WMQv7.0.1.6	152	5960	0.03	51%
WMQv7.1	136	9833	0.016	66%

Table 9 – 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. It is only the peak rate that is interesting because that is only time that the system is bottlenecking on resource contention.

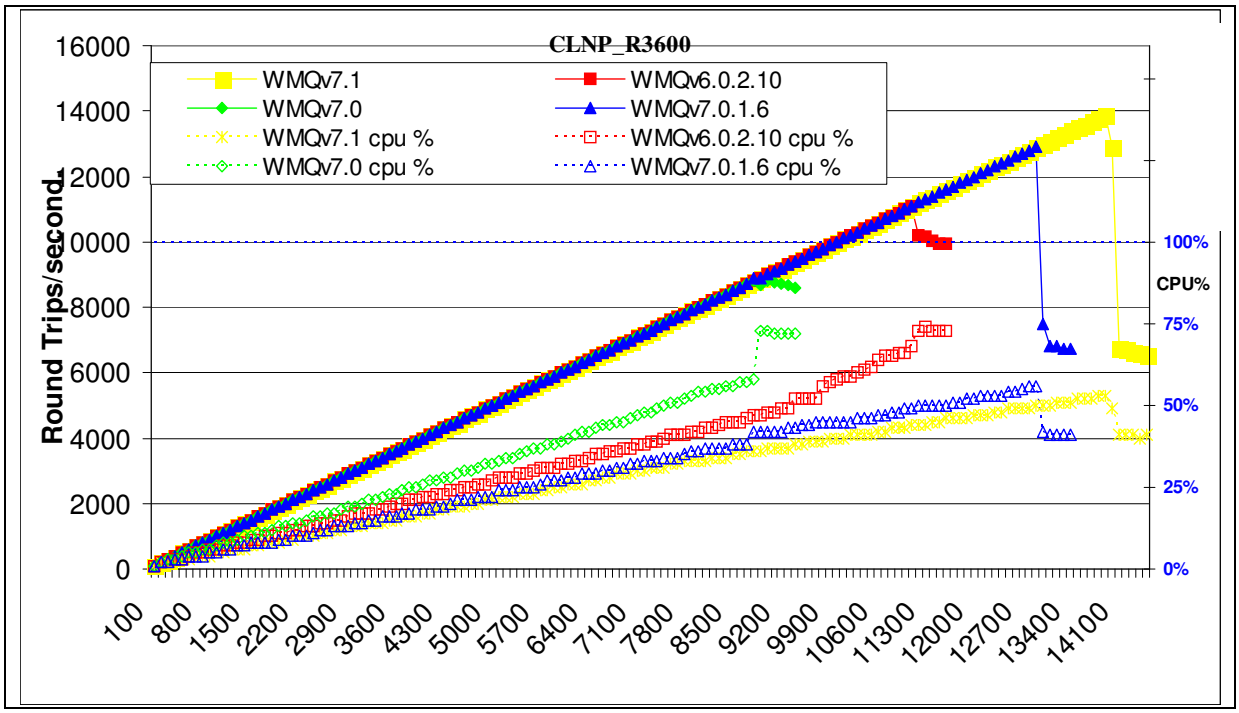


Figure 12 – 1 round trip per driving application per second, client channels and non-persistent messages

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

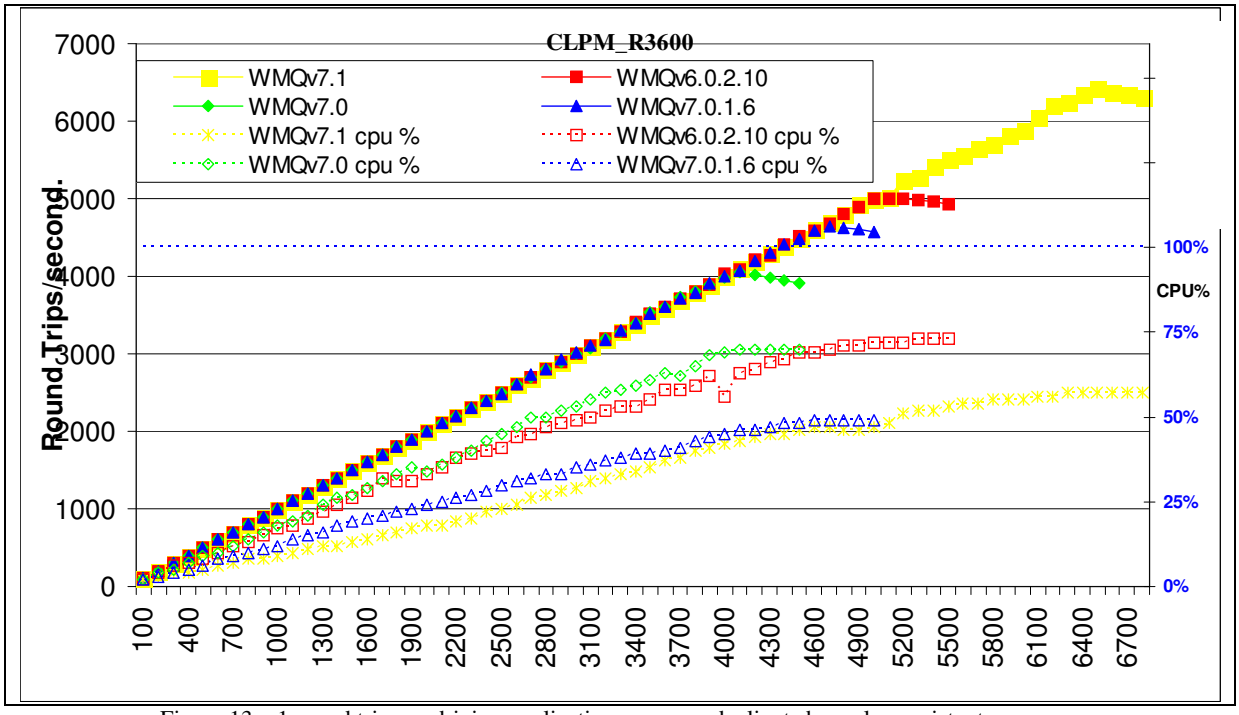


Figure 13 – 1 round trip per driving application per second, client channels, persistent messages

Figure 12, Figure 13 and Table 10 show that the throughput of non-persistent messages has increased by 55% when comparing version 7.1 to V7.0 and 8% when comparing V7.1 with V7.0.1.6. Persistent messages have improved by 58% when comparing with V7.0 and 38% compared with V7.0.1.6

Test Name: CLNP_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	11100	11099	0.27	68%
WMQv7.0	8800	8799	0.0017	58%
WMQv7.0.1.6	12900	12898	0.0072	56%
WMQv7.1	13900	13898	0.0014	53%

Test Name: CLPM_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	4900	4894	0.99	71%
WMQv7.0	3900	3890	0.42	68%
WMQv7.0.1.6	4400	4411	0.8	48%
WMQv7.1	6200	6207	0.94	56%

Table 10 – 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 Scaling – Client channels

These measurements used an LPAR on the Power 7 machine (used elsewhere in this report as the Driver system) as the Driver/Queue manager with the Power 6 system used as the driver. This application model of using a single server input Queue stresses the locking characteristics of the system and MQ V7.1 can utilise more cores than V7.0.1.6

2.2.5.1 Scaling - Client Non Persistent V7.0.1.6

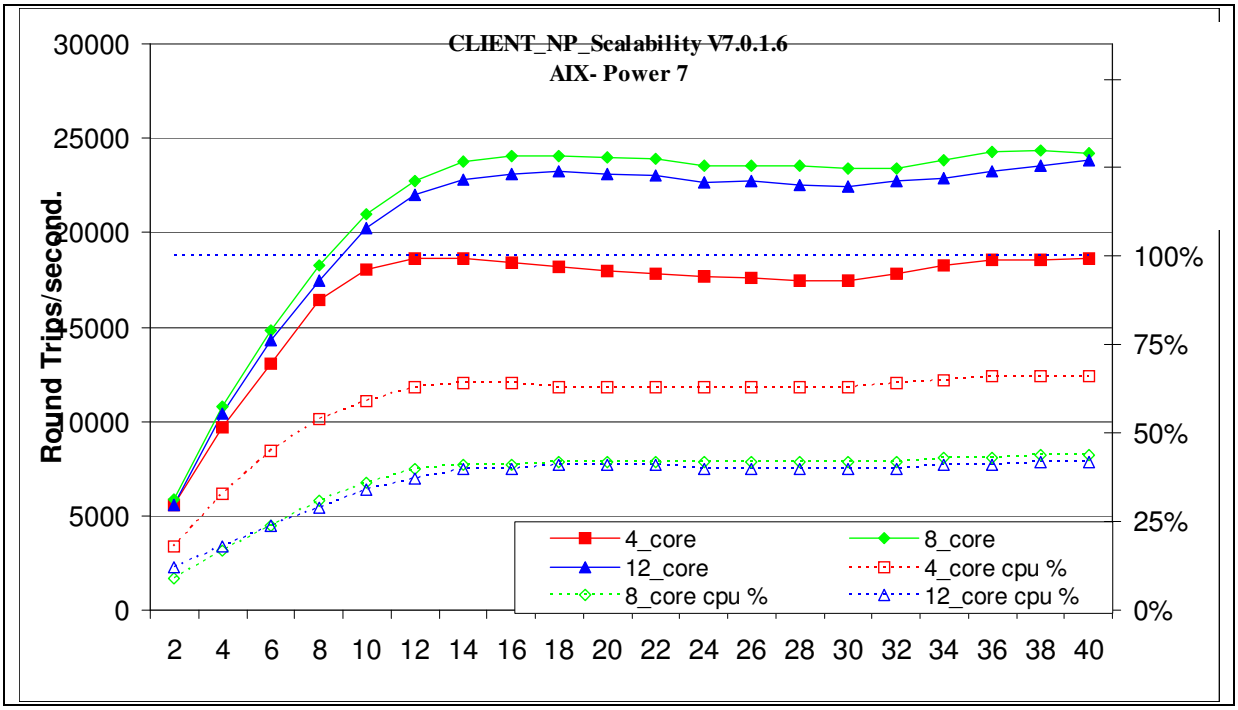


Figure 14 Scaling Client NP V7.0.1.6

Test Name: CLIENT_NP_Scalability V7.0.1.6	Apps	Round Trips/Sec	Response time (s)	CPU
4_core	14	18653	0.00093	64%
8_core	38	24320	0.0018	44%
12_core	40	23803	0.002	42%

Table 11 Scaling Client NP V7.0.1.6

Figure 14 and Table 11 show an 8 core system can process 30% more messages than a 4 core system but peak throughput is reduced when 12 cores are used.

2.2.5.2 Scaling - Client Non Persistent V7.1

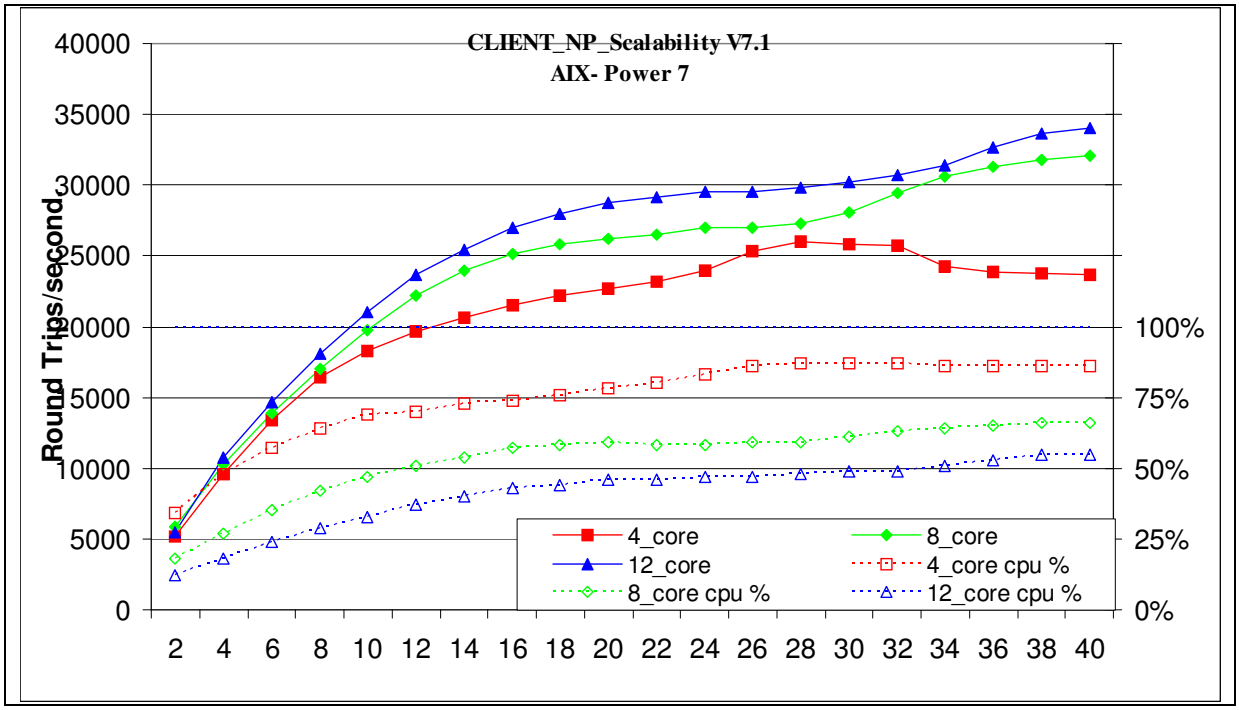


Figure 15 Scaling Client NP V7.1

Test Name: CLIENT_NP_Scalability V7.1	Apps	Round Trips/Sec	Response time (s)	CPU
4_core	28	25974	0.0013	87%
8_core	40	32034	0.0014	66%
12_core	40	34015	0.0013	55%

Table 12 Scaling Client NP V7.1

Figure 15 and Table 12 show the peak messaging rate of a 8 core system is 23% higher than a 4 core system. The peak processing rate of a 12 core system is 30% higher than a 4 core system.

These Client scalability charts show 12 cores provide an increase in throughput over 8 cores whereas in V7.0.1.6 they did not do so.

2.3 Distributed Queuing Test Scenario

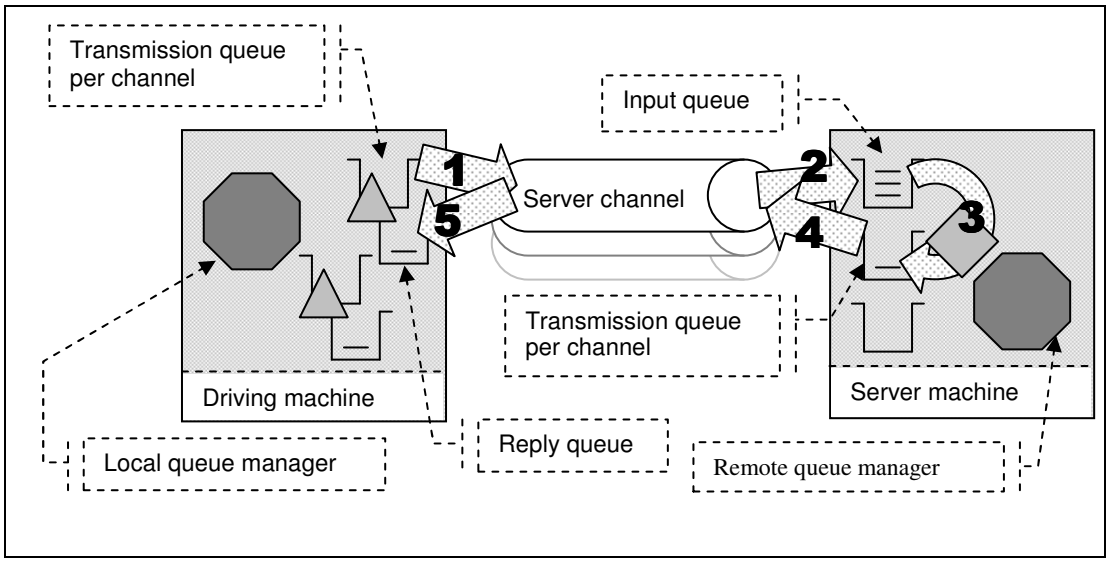


Figure 16 – Server channels between two queue managers

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 17, Figure 18 and Figure 19 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 16 on the previous page) and WebSphere MQ (versions 7.1, 7.0.1.6, 7.0 and 6.0.2.11).

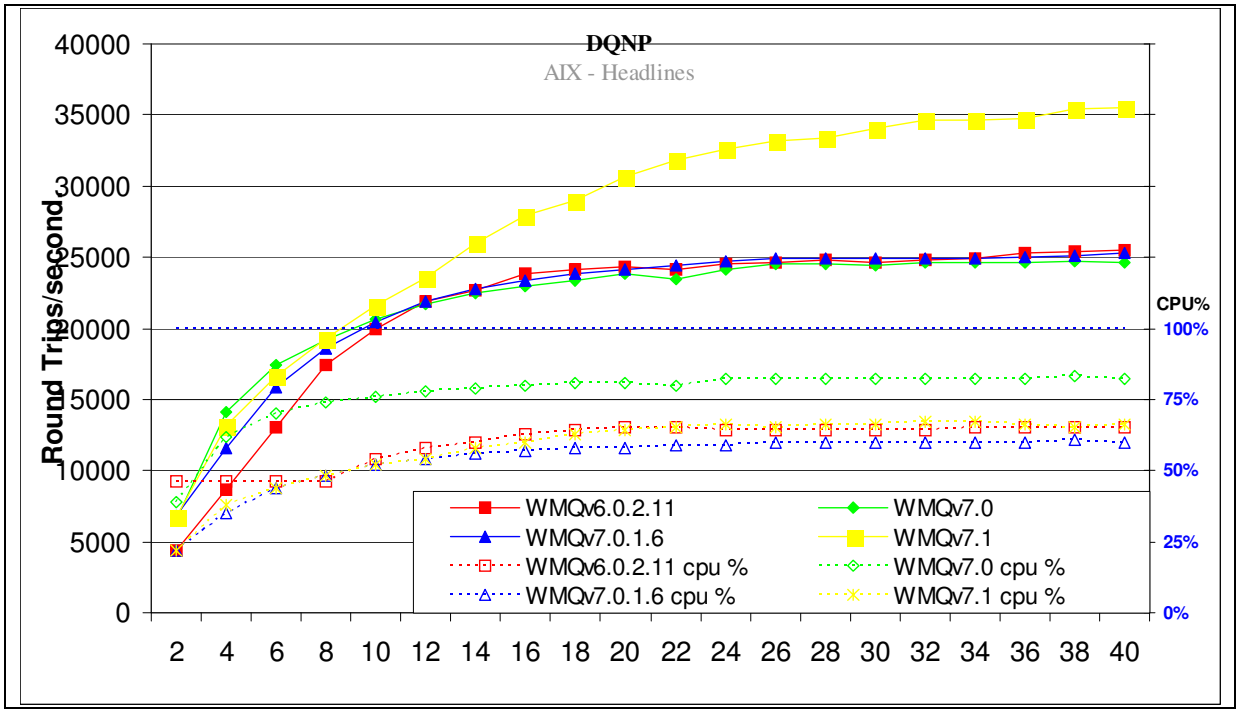


Figure 17 – Performance headline, non-persistent messages and server channels

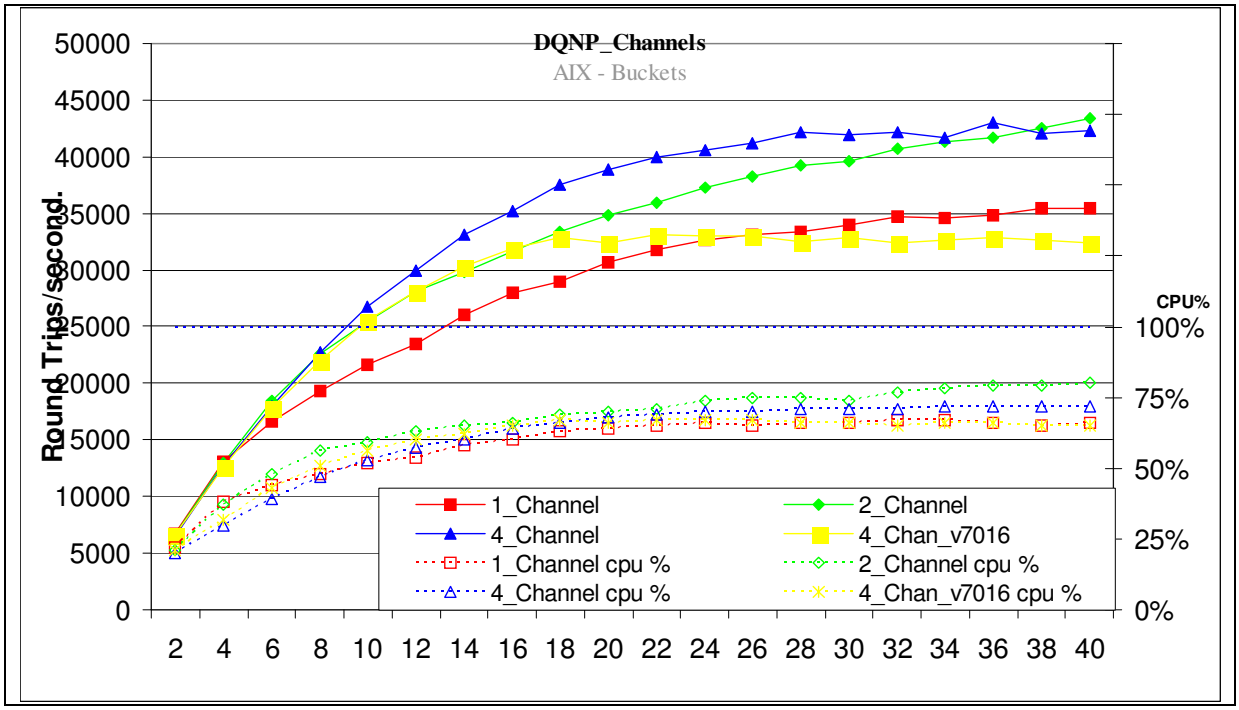
Figure 17 and Table 13 shows that the throughput of non-persistent messages has improved by 25% when comparing version 7.1 to 7.0.

Test Name: DQNP	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	40	25544	0.0018	65%
WMQv7.0	38	24723	0.0019	83%
WMQv7.0.1.6	40	25316	0.0019	60%
WMQv7.1	40	35492	0.0013	66%

Table 13 – 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

2.3.1.1 Non persistent – Multiple Server Channels



Test Name: DQNP_Channels	Apps	Round Trips/Sec	Response time (s)	CPU
1_Channel	40	35492	0.0013	66%
2_Channel	40	43356	0.001	80%
4_Channel	36	43078	0.00095	72%
4_Chan_v7016	22	33147	0.00076	67%

The limiting factor of this model is the single pairs of channels. By ensuring the applications are evenly distributed over 2 or 4 channels, the throughput can be increased significantly. 2 channels shows a 16% increase over a single channel while 4 channels increase throughput by a further 5% in Version 7.1

It can also be seen that the performance advantages of using 4 channels on V7.1 is 16 % higher throughput than V7.0.1.6

2.3.2 Non-Persistent non-Trusted – Server Channels

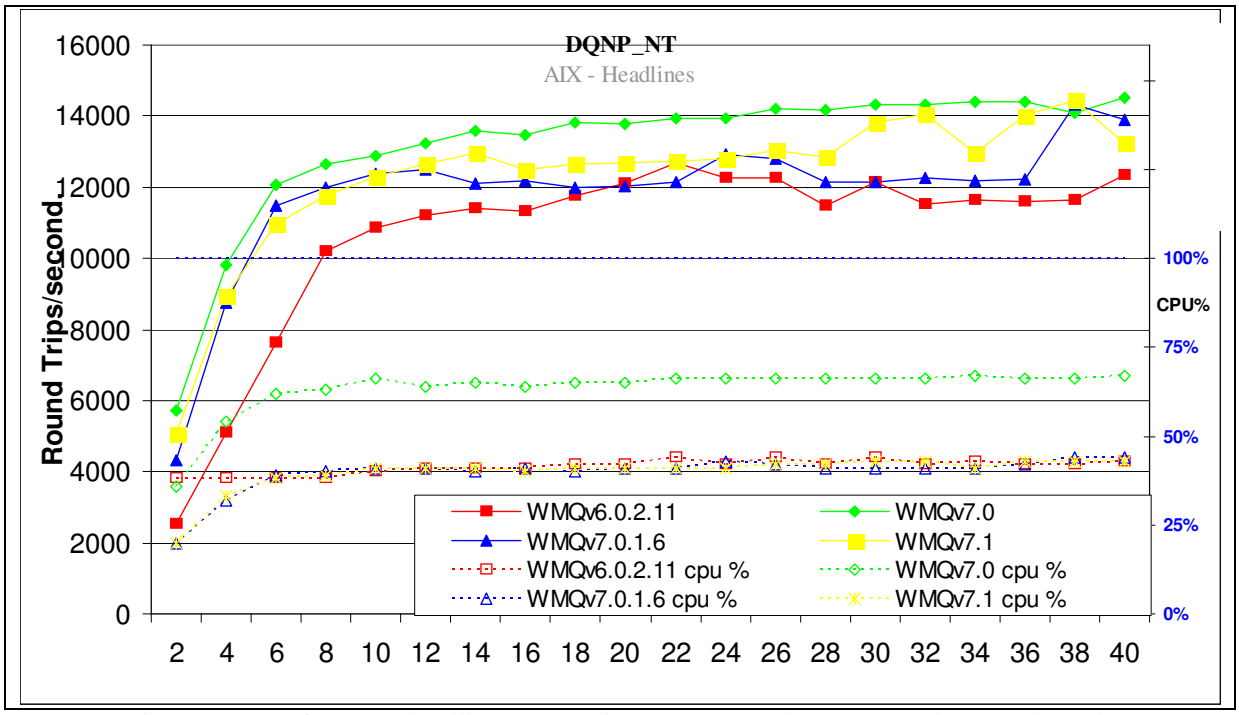


Figure 18 – Performance headline, non-persistent, not trusted messages and server channels

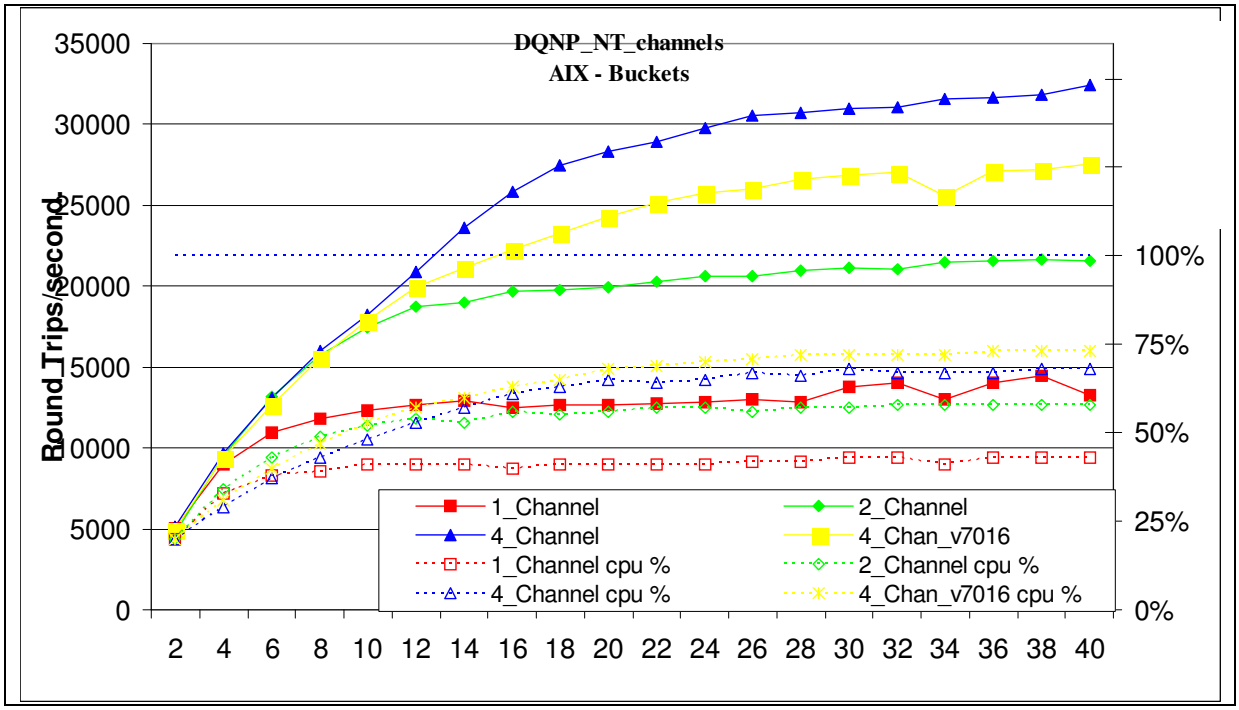
Error! Not a valid link. and **Error! Not a valid link.** shows that the throughput **Error! Not a valid link.** of non-persistent, non-trusted messages has decreased by 9% when comparing version 7.1 to 7.0 but the cpu cost per message has improved by 33%.

Test Name: DQNP_NT	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	22	12674	0.0021	44%
WMQv7.0	40	14505	0.0034	67%
WMQv7.0.1.6	38	14309	0.0032	44%
WMQv7.1	38	14435	0.0036	43%

Table 14 – 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

2.3.2.1 Non persistent non Trusted – Multiple Server Channels



Test Name: DQNP_NT_channels	Apps	Round Trips/Sec	Response time (s)	CPU
1_Channel	38	14435	0.0036	43%
2_Channel	38	21643	0.0023	58%
4_Channel	40	32462	0.0014	68%
4_Chan_v7016	40	27554	0.0017	73%

The limiting factor of this model is the single pair of channels. By ensuring the applications are evenly distributed over 2 or 4 channels, the throughput can be increased significantly. 2 channels show an increase in throughput of 60% over 1 channel and 4 channels show an increase of 50% over 2 channels.

It can also be seen that the performance advantages of using 4 channels on V7.1 is 17% higher throughput than V7.0.1.6

2.3.3 Persistent Messages – Server Channels

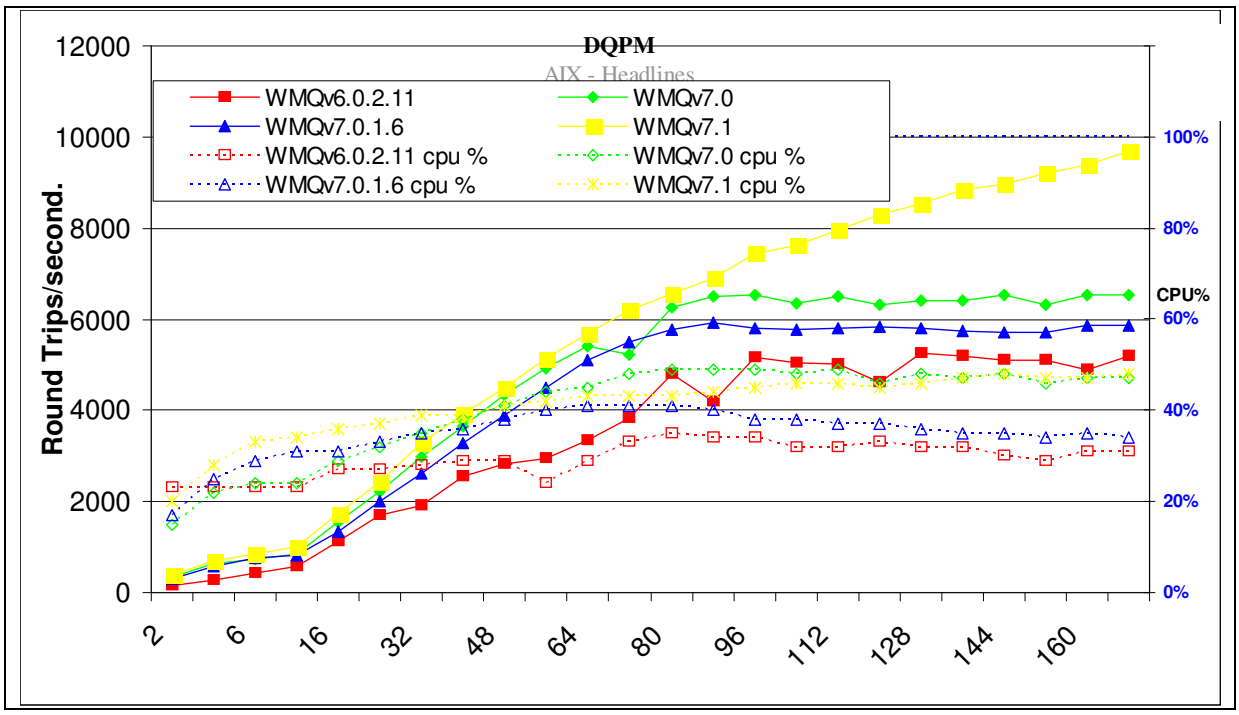


Figure 19 – Performance headline, persistent messages and server channels

Figure 19 and Table 15 shows that the throughput of persistent messages using 2 pairs of channels has increased by 15% when comparing version 7.1 to V7.0.

Test Name: DQPM	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.11	128	5248	0.04	32%
WMQv7.0	160	6542	0.029	47%
WMQv7.0.1.6	88	5909	0.017	40%
WMQv7.1	168	9678	0.022	48%

Table 15 – 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.

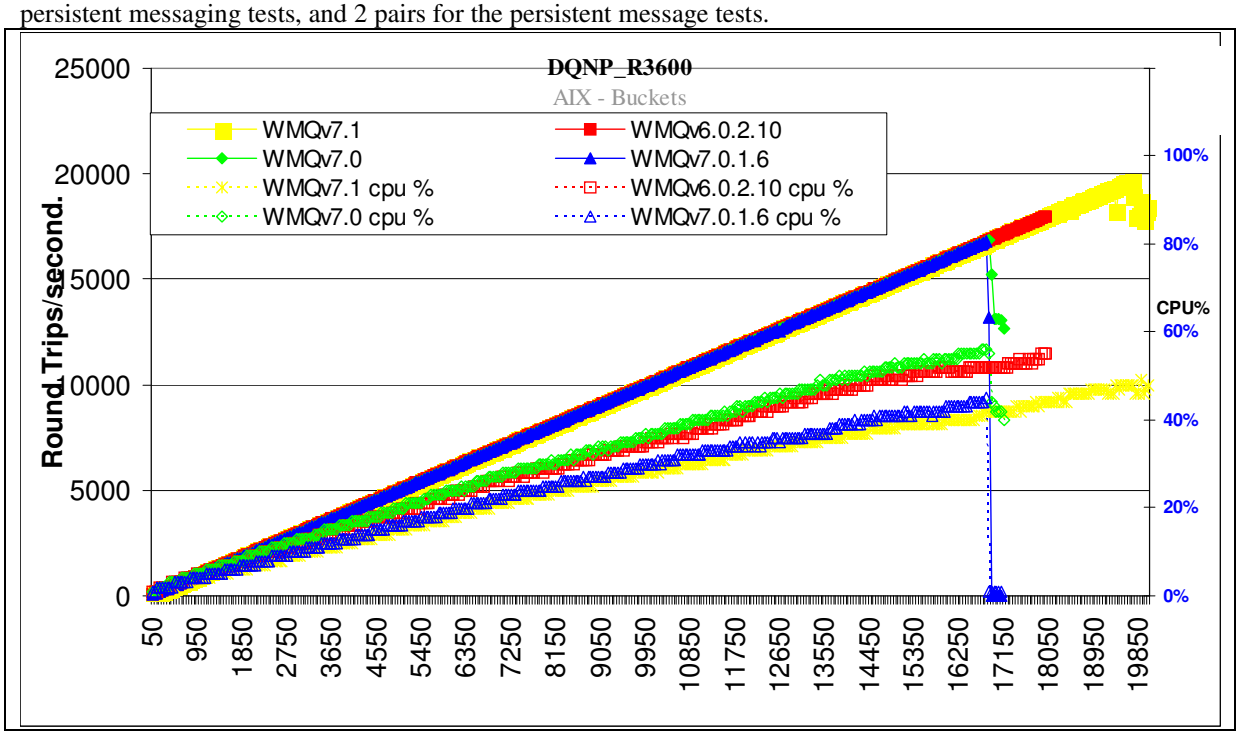


Figure 20 – 1 round trip per driving application per second, server channel, non-persistent messages

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

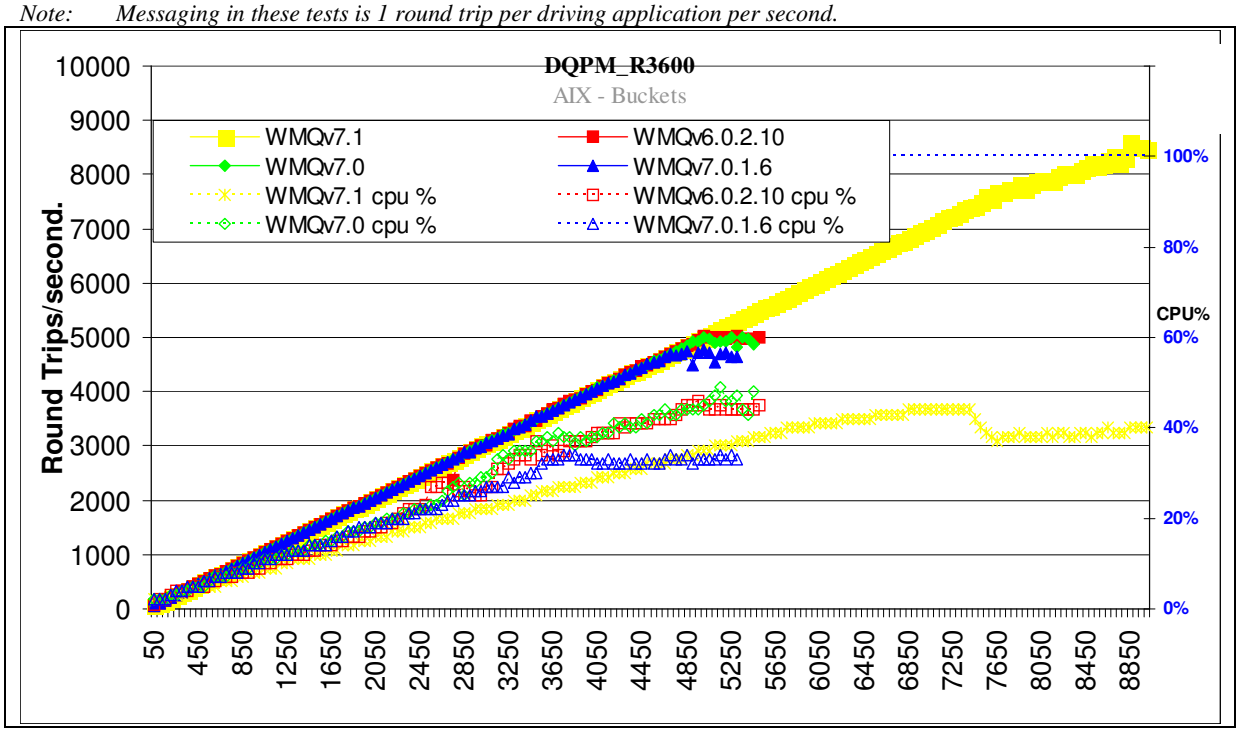


Figure 21 – 1 round trip per driving application per second, server channel, persistent messages

Figure 20, Figure 21 and Table 16 shows that the throughput of non-persistent messages has improved by 17% for non Persistent and 70% for Persistent when comparing version 7.1 to 7.0.

Test Name: DQNP_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	18000	18001	0.0035	55%

WMQv7.0	16850	16847	0.0025	55%
WMQv7.0.1.6	16800	16794	0.00077	45%
WMQv7.1	19700	19698	0.00081	48%

Test Name: DQPM_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	5000	5035	0.2	45%
WMQv7.0	5000	5005	0.064	45%
WMQv7.0.1.6	5000	4765	0.83	33%
WMQv7.1	8850	8606	0.92	40%

Table 16 – 1 round trip per driving application per second, client channels

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

3 Large Messages

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

3.1.1 50bytes to 32KB

Figure 22 show the response time for MQPut/MQGet for NP message sizes between 50bytes and 32Kb.

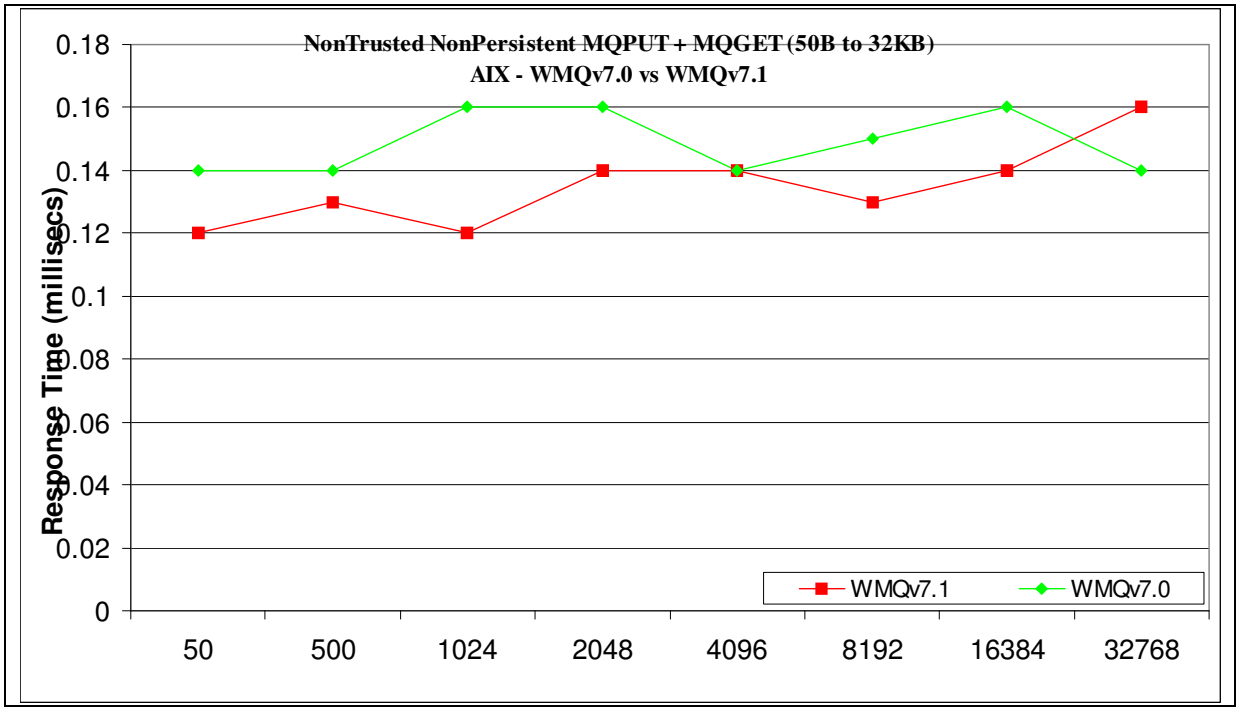


Figure 22 –The effect of non-persistent message size on MQI response time (50byte - 32K)

Figure 23 show that the response for MQPut/MQGet pairs for pers message sizes between 50bytes and 32Kb.

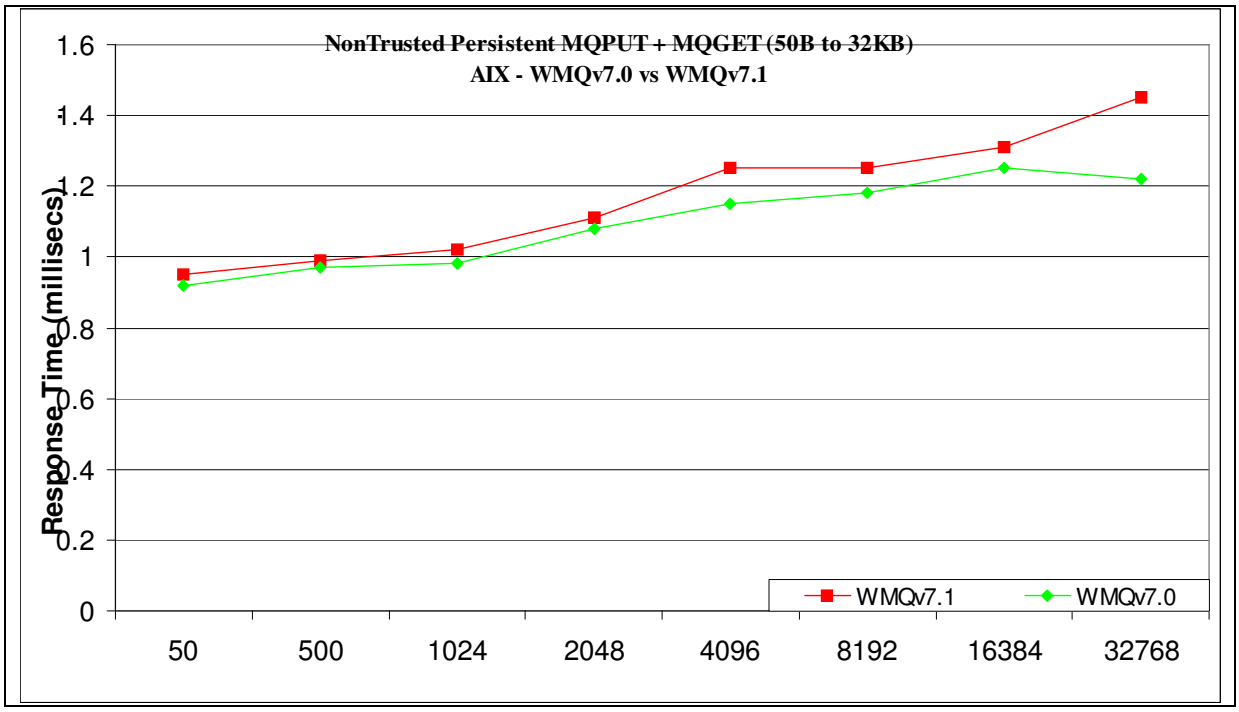


Figure 23 –The effect of persistent message size on MQI response time (50byte - 32K)

3.1.2 32KB to 2MB

Figure 24 show that the response time for MQPut/MQGet pairs for non-persistent message sizes between 32KB and 2MB.

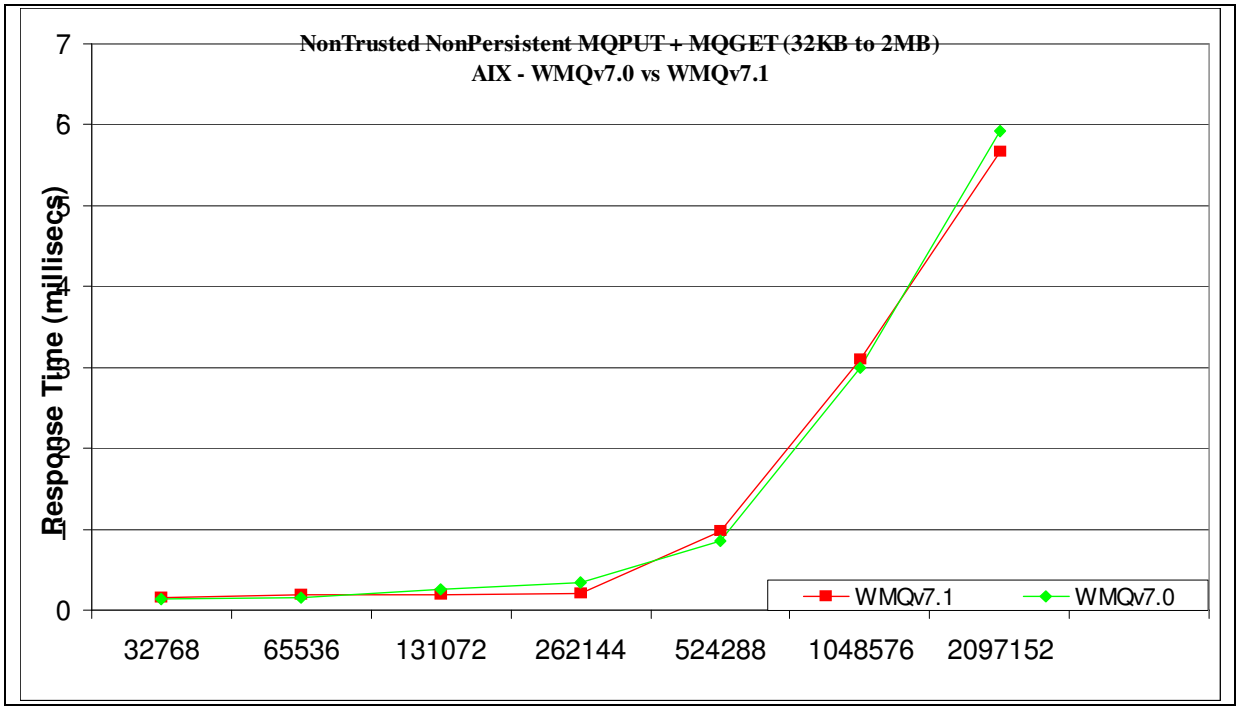


Figure 24 –The effect of non-persistent message size on MQI response time (32KB – 2MB)

Figure 25 show that the response for MQPut/MQGet pairs for persistent message sizes between 32KB and 2MB.

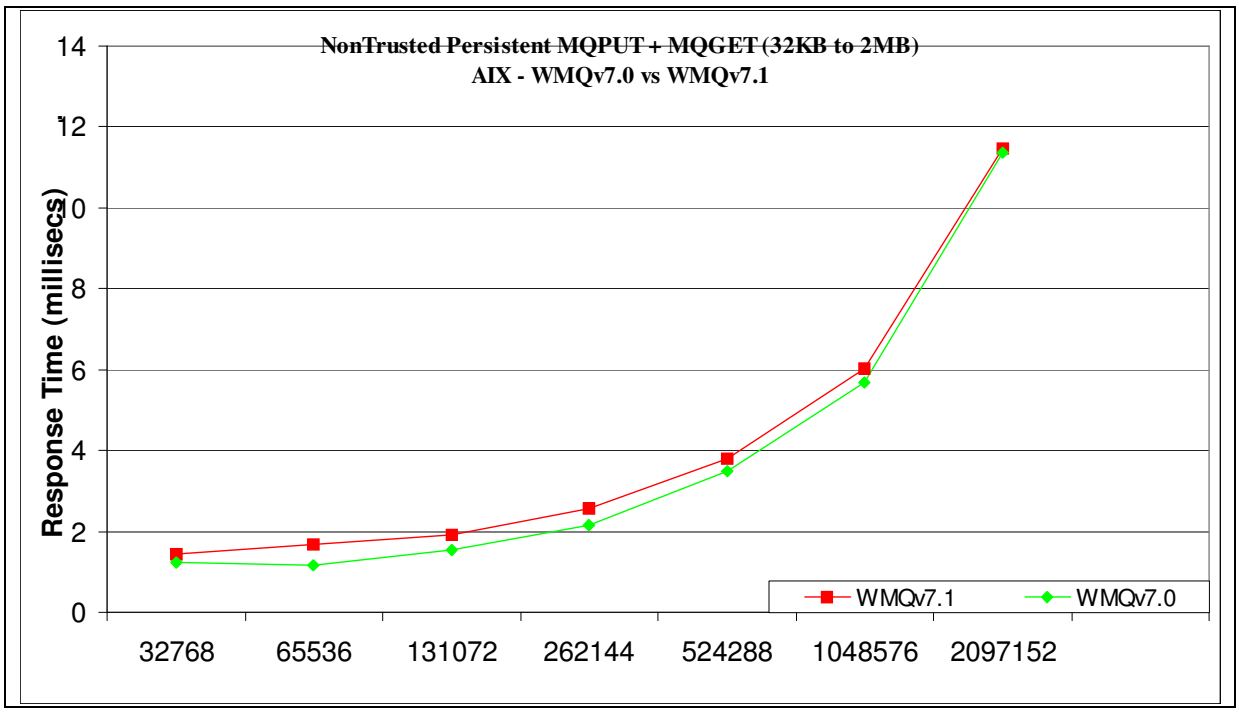


Figure 25 –The effect of persistent message size on MQI response time (32KB – 2MB)

3.1.3 2MB to 100MB

Figure 26 Response time for MQPut/MQGet pairs for NP message between 2MB and 100MB.

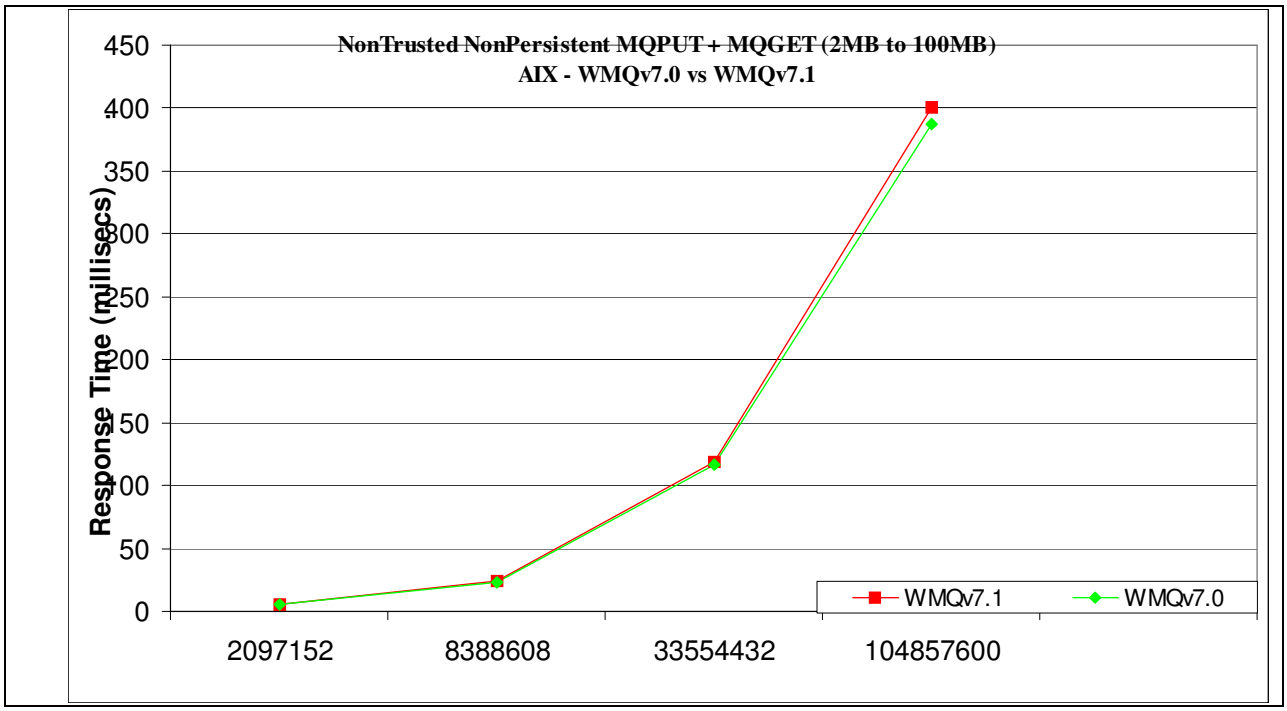


Figure 26 –The effect of non-persistent message size on MQI response time (2MB – 100MB)

Figure 27 The response for MQPut/MQGet pairs for persistent message sizes between 2MB and 33MB.

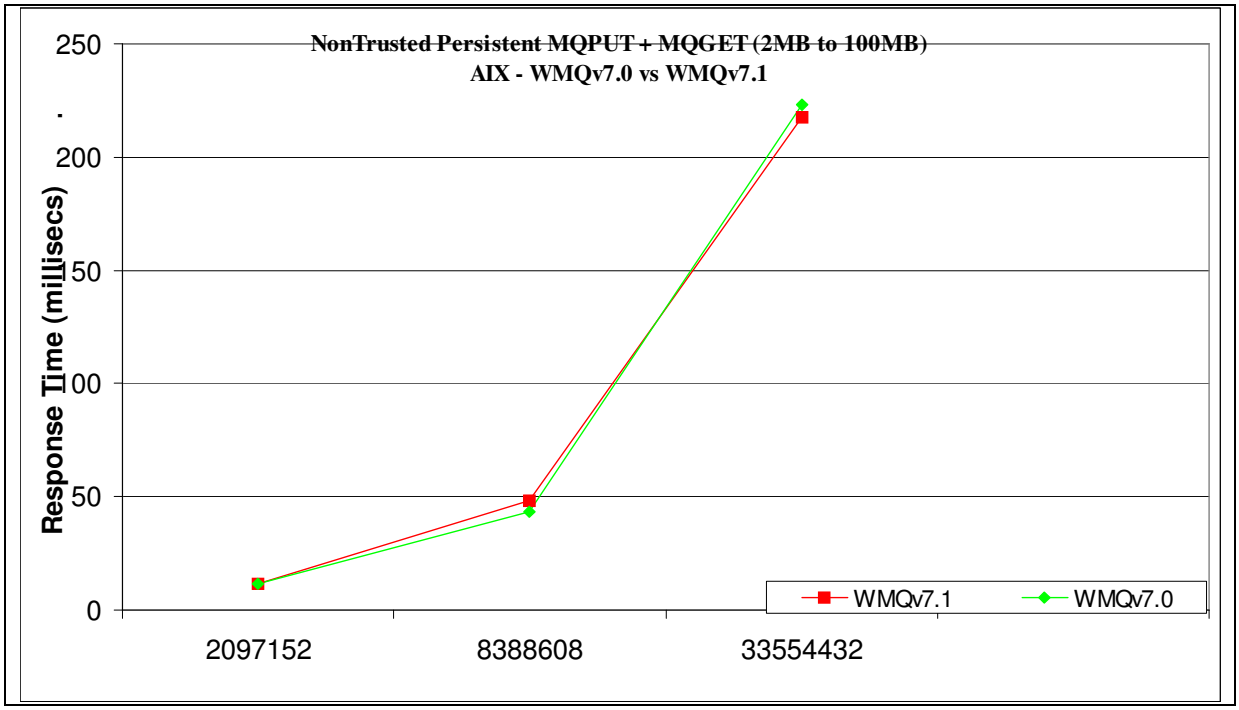


Figure 27 –The effect of persistent message size on MQI response time (2MB – 33MB)

3.2 20KB Messages

3.2.1 Local Queue Manager

Figure 28 and Figure 29 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

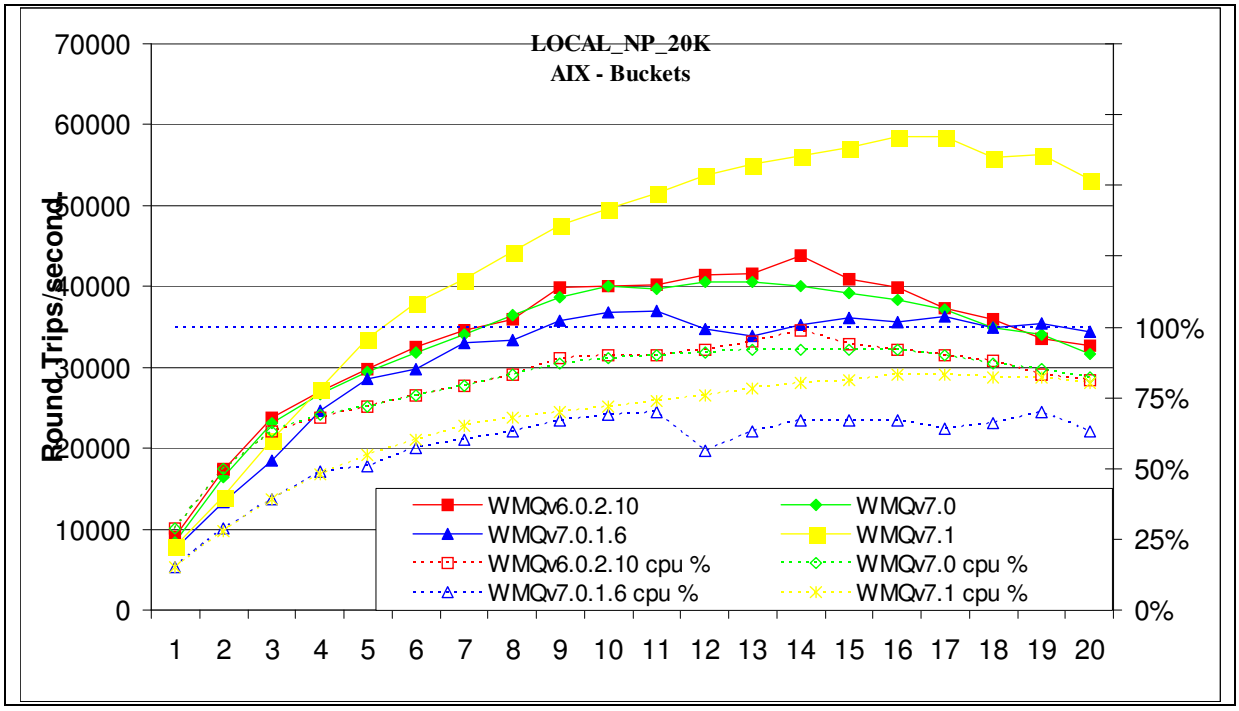


Figure 28 – 20KB non-persistent messages, local queue manager

Figure 28 and Table 17 shows that the throughput of non-persistent messages has increased by 33% when comparing version 7.1 to V7.0.

Test Name: LOCAL_NP_20K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	14	43818	0.0004	99%
WMQv7.0	13	40518	0.0004	92%
WMQv7.0.1.6	11	37031	0.00025	70%
WMQv7.1	16	58570	0.00034	83%

Table 17 – 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

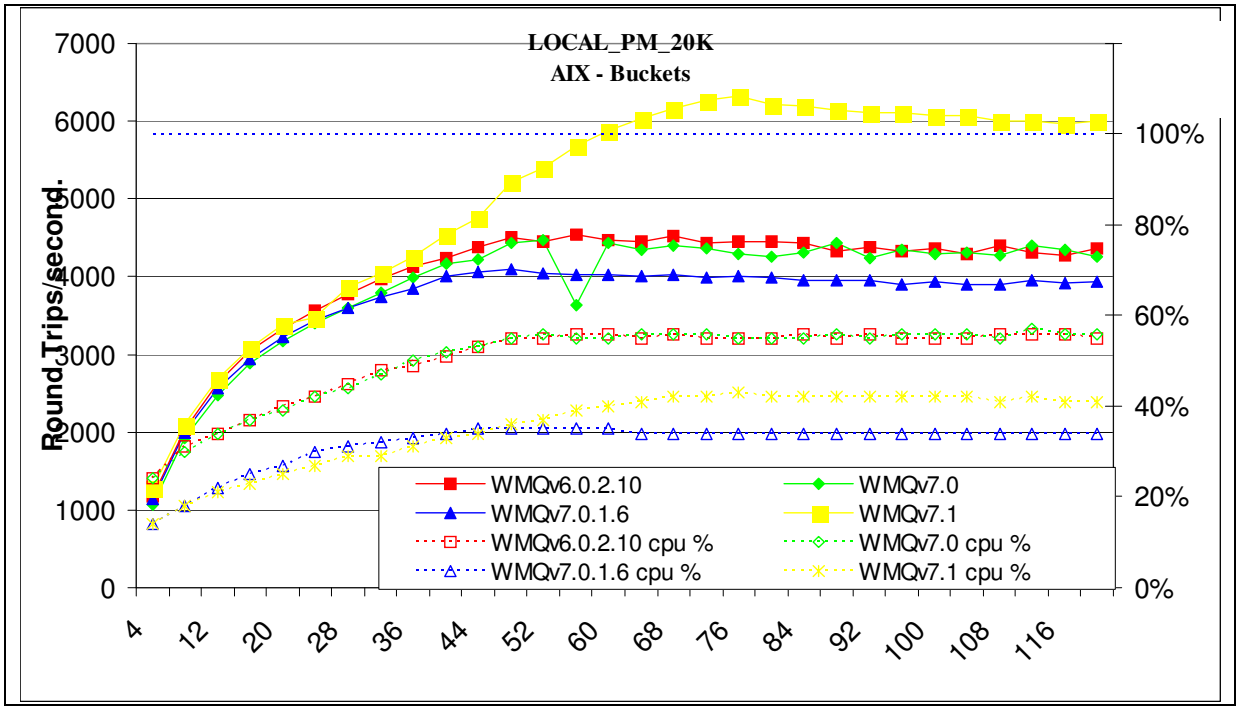


Figure 29 – 20KB persistent messages, local queue manager

Figure 29 and Table 18 shows that the throughput of persistent messages has increased by 30% when comparing version 7.1 to V7.0.

Test Name: LOCAL_PM_20K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	56	4534	0.013	56%
WMQv7.0	52	4480	0.014	56%
WMQv7.0.1.6	48	4093	0.013	35%
WMQv7.1	76	6319	0.014	43%

Table 18 – 20KB persistent messages, local queue manager

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

3.2.2 Client Channel

Figure 30 and Figure 31 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

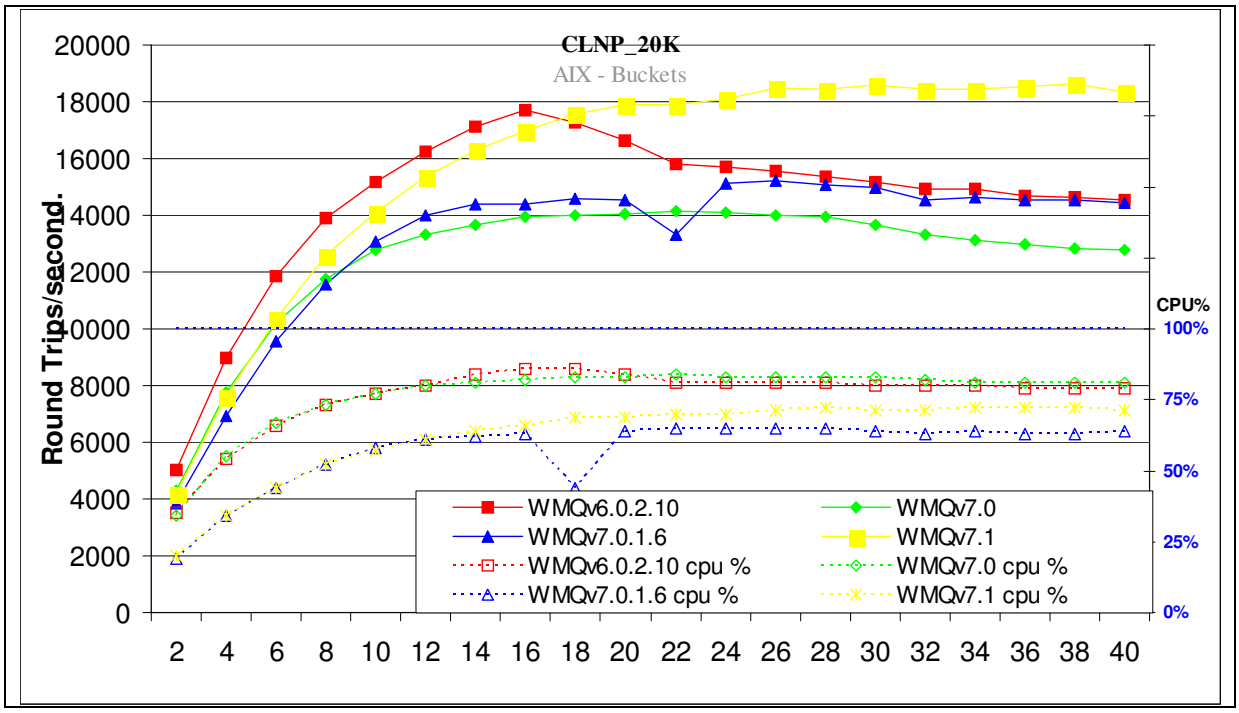


Figure 30 – 20KB non-persistent messages, client channels

Figure 30 and Table 19 shows that the throughput of non-persistent messages has increased by 26% when comparing version 7.1 to V7.0 and by 9% when comparing with v6.0.2.10

Test Name: CLNP_20K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	16	17704	0.00099	86%
WMQv7.0	22	14149	0.0018	84%
WMQv7.0.1.6	26	15209	0.0019	65%
WMQv7.1	38	18657	0.0023	72%

Table 19 – 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

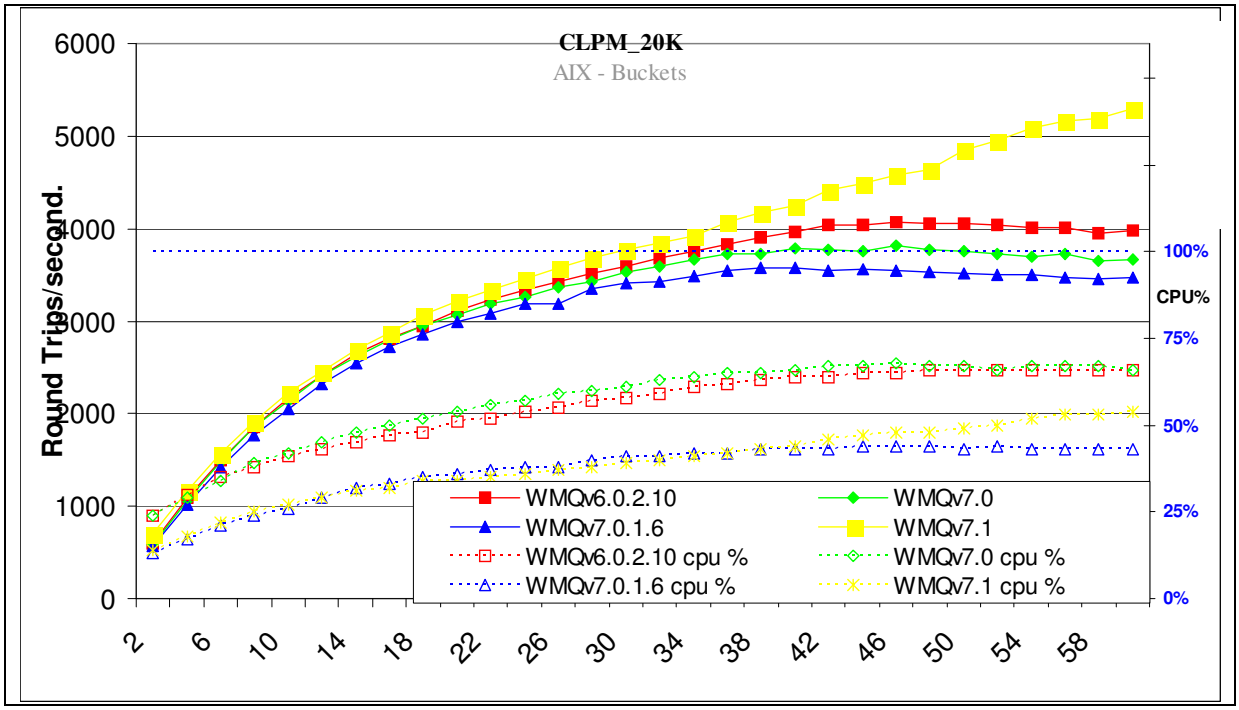


Figure 31 – 20KB persistent messages, client channels

Figure 31 and Table 20 shows that the throughput of persistent messages has increased by 16% when comparing version 7.1 to V7.0 and by 11% when compared to v6.0.2.10.

Test Name: CLPM_20K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	46	4076	0.013	65%
WMQv7.0	46	3812	0.014	68%
WMQv7.0.1.6	40	3575	0.013	43%
WMQv7.1	60	5294	0.013	54%

Table 20 – 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 32 and Figure 33 shows 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

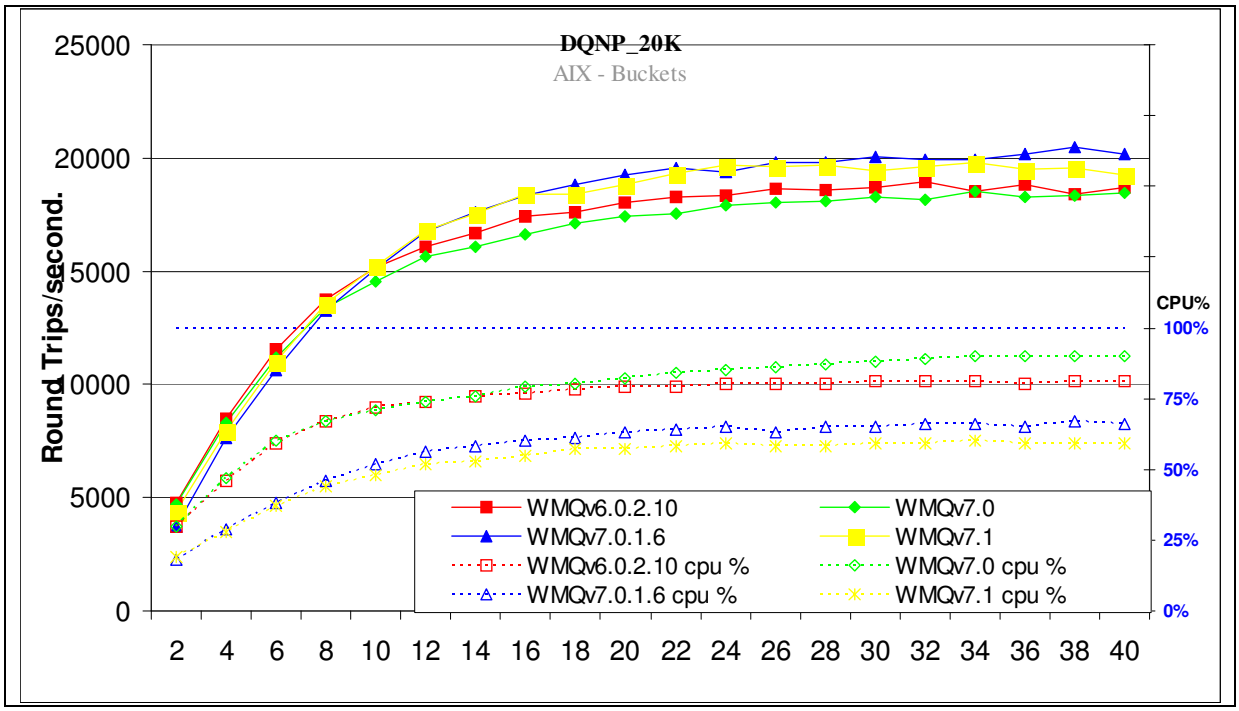


Figure 32 – 20KB non-persistent messages, distributed queuing

Figure 32 and Table 21 shows that the throughput of non-persistent messages has increased by 7% when comparing version 7.1 to V7.0.

Test Name: DQNP_20K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	32	18925	0.002	81%
WMQv7.0	34	18551	0.002	90%
WMQv7.0.1.6	38	20479	0.0019	67%
WMQv7.1	34	19792	0.002	60%

Table 21 – 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

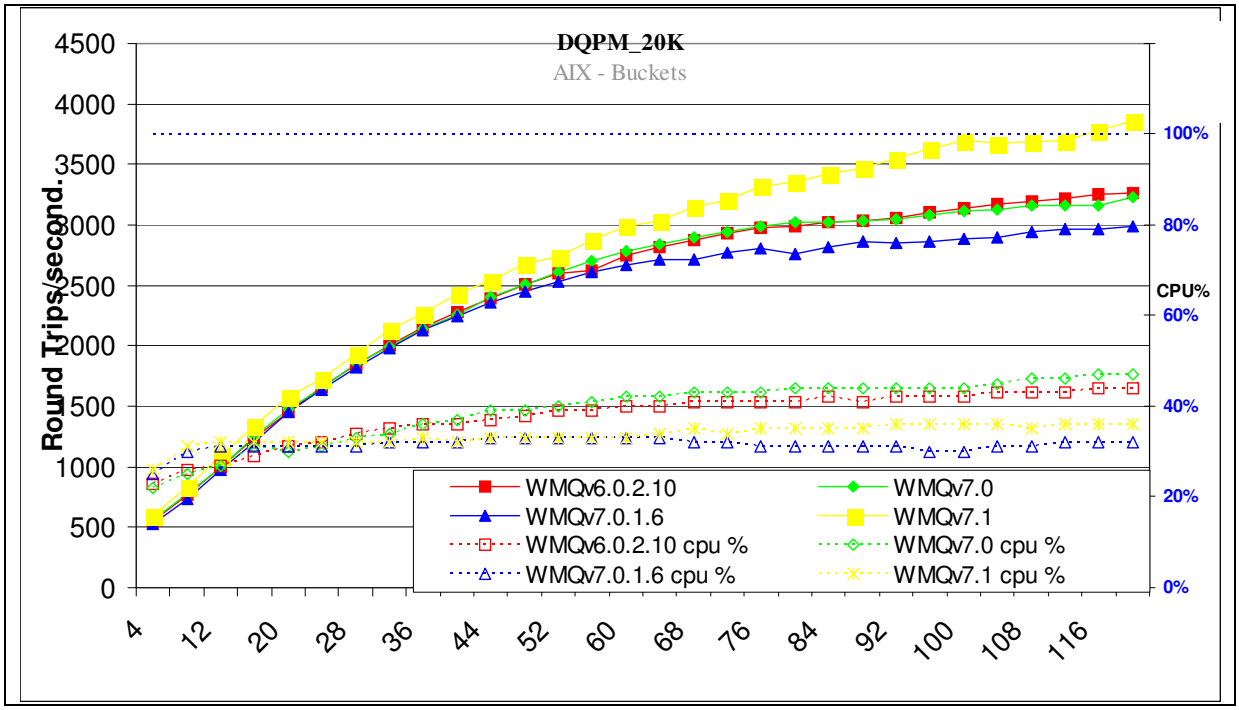


Figure 33 – 20KB persistent messages, distributed queuing

Figure 33 and Table 22 shows that the throughput of persistent messages has increased by 12% when comparing version 7.1 to V7.0.

Test Name: DQPM_20K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	120	3265	0.051	44%
WMQv7.0	120	3233	0.039	47%
WMQv7.0.1.6	120	2992	0.059	32%
WMQv7.1	120	3861	0.033	36%

Table 22 – 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 34 and Figure 35 shows 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

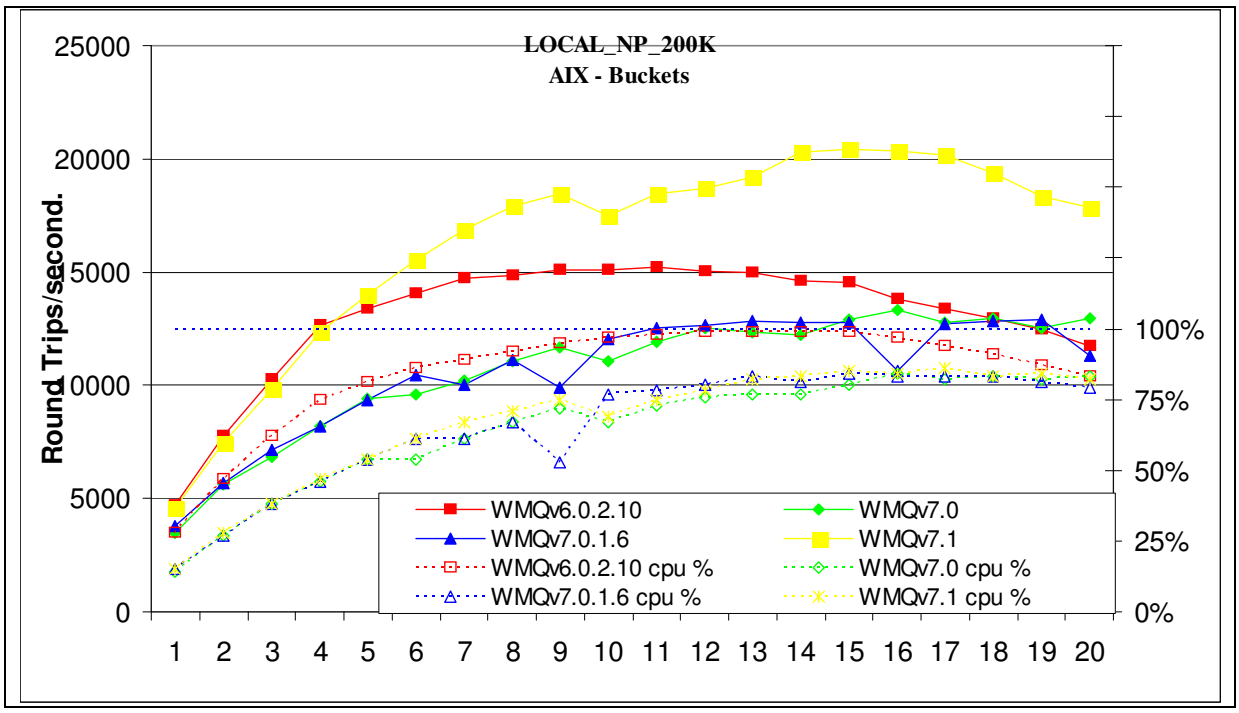


Figure 34 – 200KB non-persistent messages, local queue manager

Figure 34 and Table 23 shows that the throughput of non-persistent messages has increased by 50% when comparing version 7.1 to V7.0.

Test Name: LOCAL_NP_200K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	11	15193	0.00089	98%
WMQv7.0	16	13300	0.00095	84%
WMQv7.0.1.6	19	12925	0.00098	81%
WMQv7.1	15	20442	0.00075	85%

Table 23 – 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

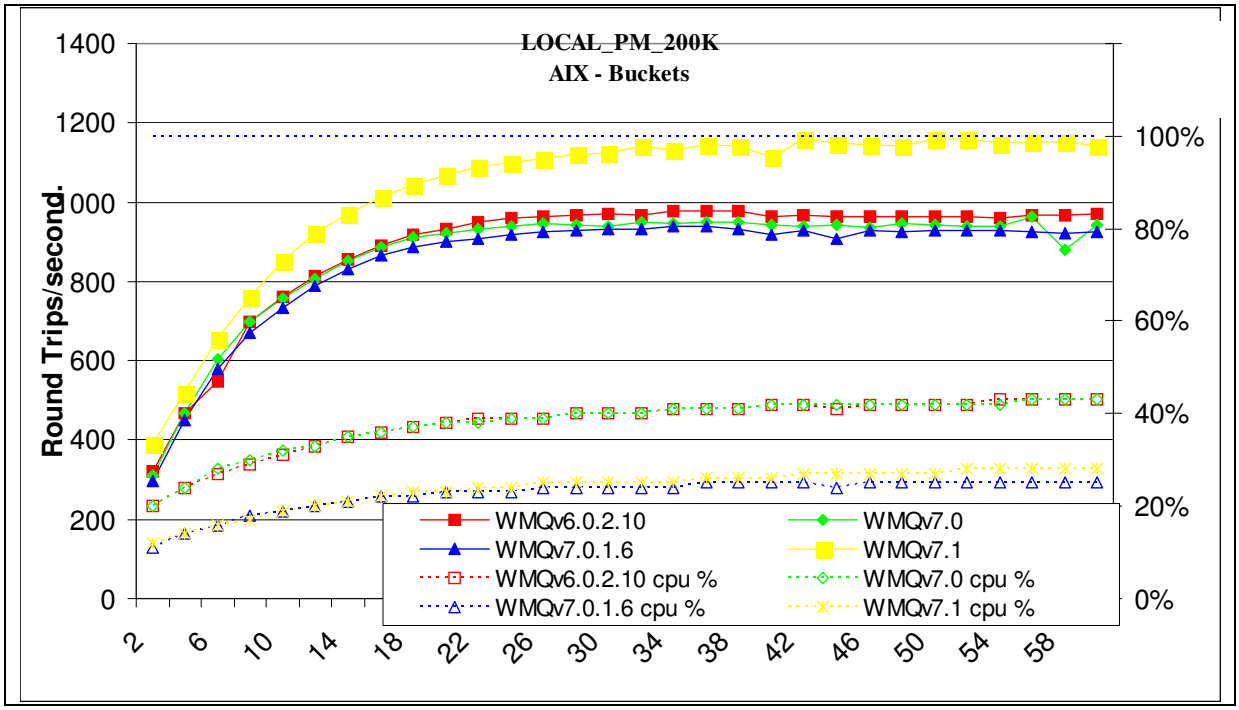


Figure 35 – 200KB persistent messages, local queue manager

Figure 35 and Table 24 shows that the throughput of persistent messages has increased by 19% when comparing version 7.1 to V7.0.

Test Name: LOCAL_PM_200K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	38	978	0.045	41%
WMQv7.0	56	962	0.067	43%
WMQv7.0.1.6	34	938	0.04	24%
WMQv7.1	52	1159	0.052	28%

Table 24 – 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 36 and Figure 37 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

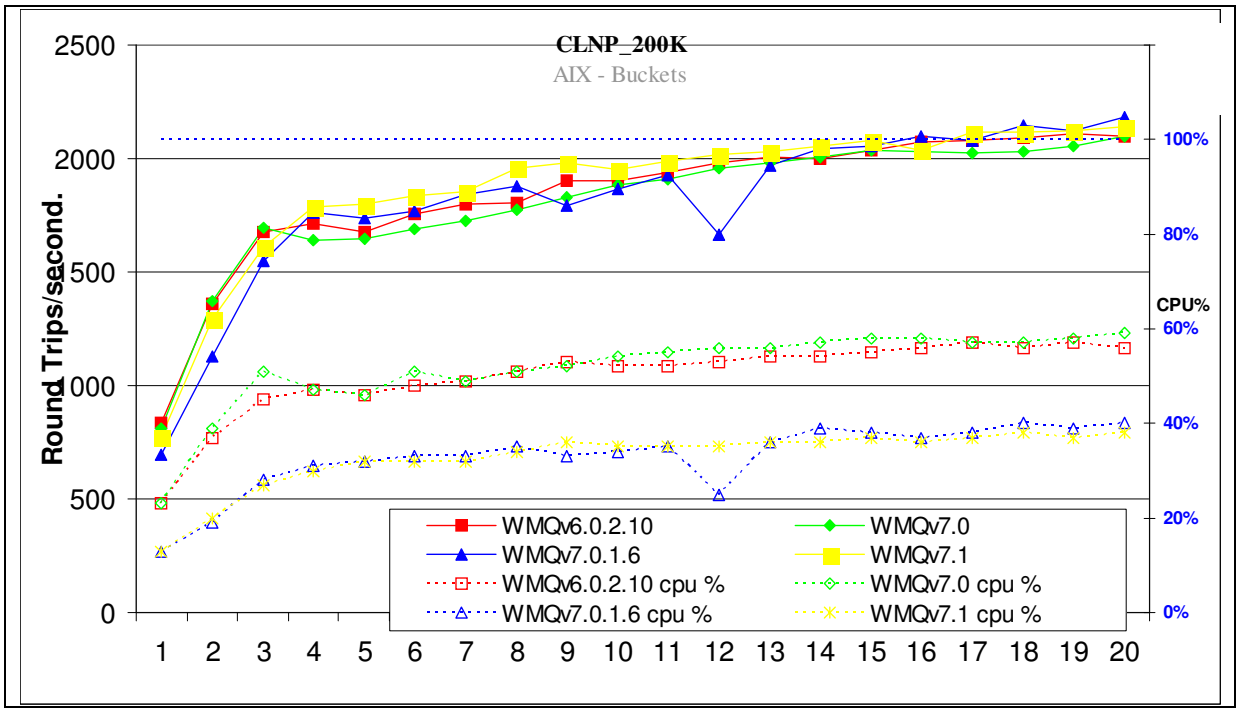


Figure 36 – 200KB non-persistent messages, client channels

Figure 36 and Table 25 shows that the throughput of non-persistent messages has increased by 4% when comparing version 7.1 to V7.0.

Test Name: CLNP_200K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	19	2107	0.0025	57%
WMQv7.0	20	2095	0.0028	59%
WMQv7.0.1.6	20	2182	0.035	40%
WMQv7.1	20	2140	0.0054	38%

Table 25 – 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

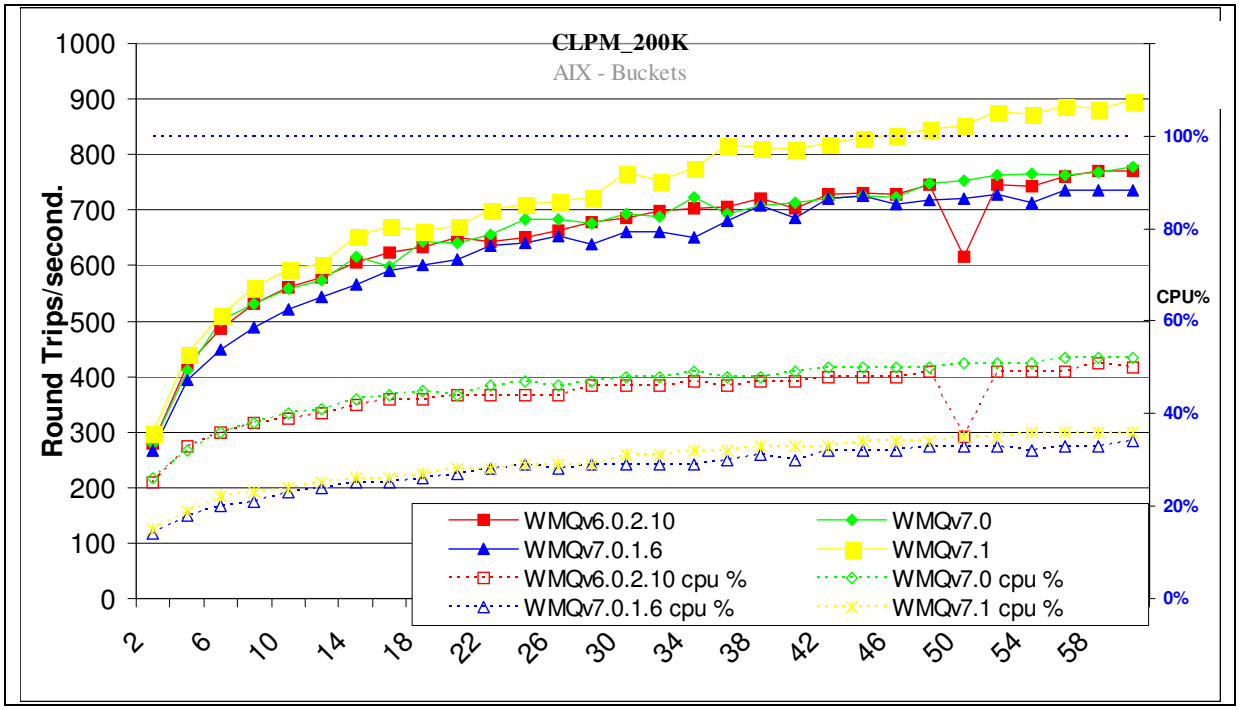


Figure 37 – 200KB persistent messages, client channels

Figure 37 and Table 26 shows that the throughput of persistent messages has increased by 10% when comparing version 7.1 to V7.0.

Test Name: CLPM_200K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	60	771	0.081	50%
WMQv7.0	60	778	0.11	52%
WMQv7.0.1.6	58	736	0.064	33%
WMQv7.1	60	898	0.11	36%

Table 26 – 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 38 and Figure 40 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the distributed queuing scenario

3.3.3.1 Non-persistent Messages

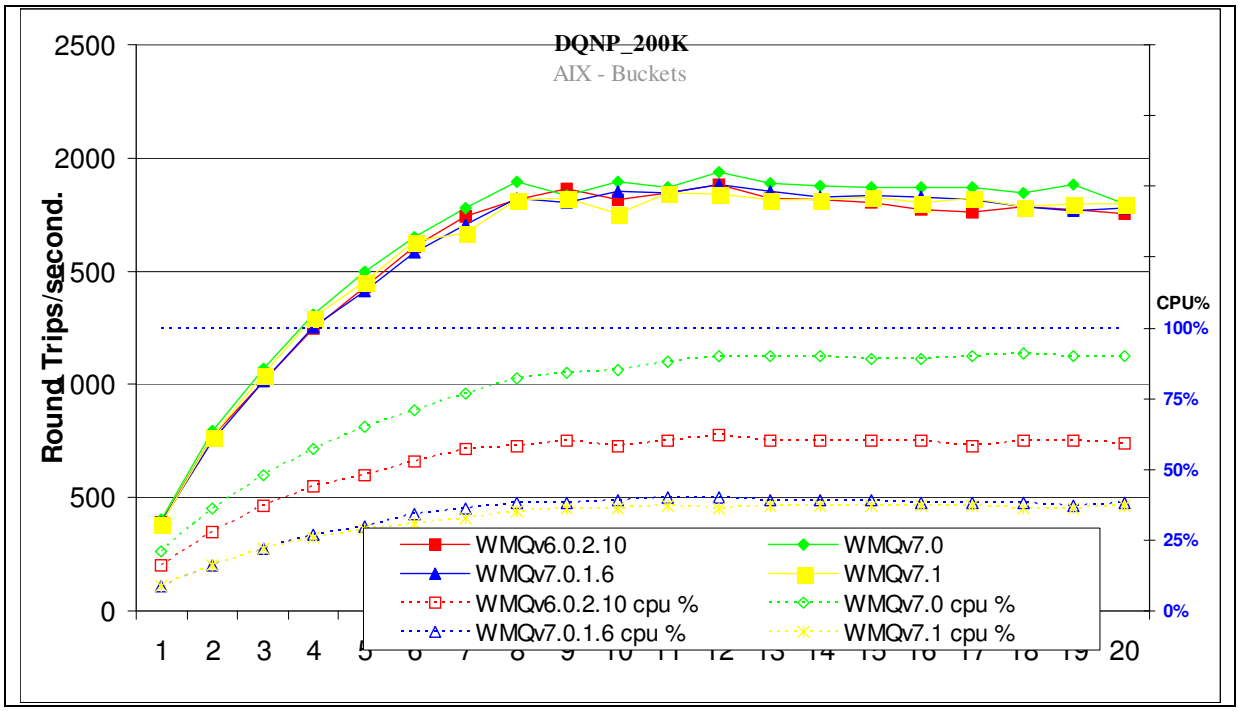


Figure 38 – 200KB non-persistent messages, distributed queuing

Figure 38 and Table 27 shows that the throughput of non-persistent messages is similar when comparing version 7.1 to V7.0.

Test Name: DQNP_200K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	12	1885	0.0065	62%
WMQv7.0	12	1940	0.0066	90%
WMQv7.0.1.6	12	1881	0.0061	40%
WMQv7.1	11	1848	0.01	37%

Table 27 – 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 Non persistent 200k – Multiple Server Channels

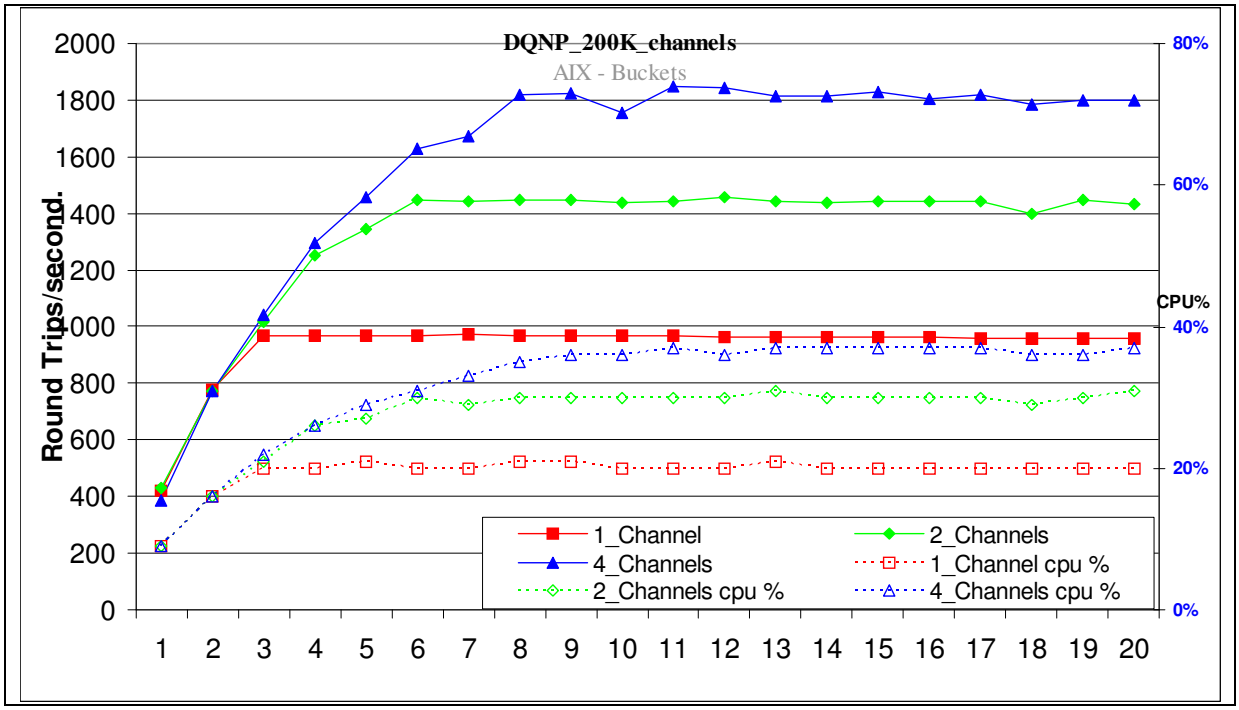


Figure 39 - 200KB non Persistent messages, Multiple Channels

Test Name: DQNP_200K_channels	Apps	Round Trips/Sec	Response time (s)	CPU
1_Channel	7	971	0.0083	20%
2_Channels	12	1456	0.0096	30%
4_Channels	11	1848	0.01	37%

Table 28 - 200KB non Persistent messages, Multiple Channels

Figure 39 and Table 28 shsow 2 channels pairs increase the throughput by 50% and using 4 channel pairs provide a 90% increase over using a single channel pair

3.3.3.3 Persistent Messages

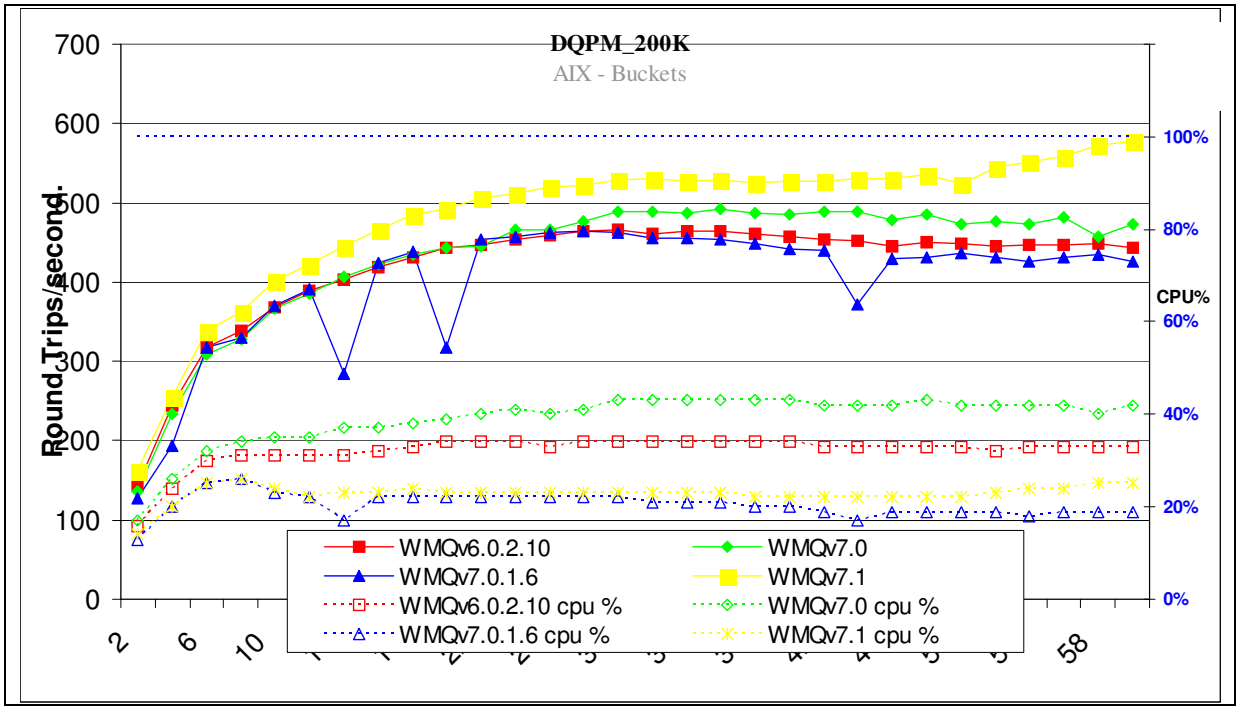


Figure 40 – 200KB persistent messages, distributed queuing

Figure 40 and Table 29 shows that the throughput of persistent messages has increased by 11% when comparing version 7.1 to V7.0.

Test Name: DQPM_200K	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	30	466	0.077	34%
WMQv7.0	36	492	0.088	43%
WMQv7.0.1.6	28	464	0.069	22%
WMQv7.1	60	578	0.12	25%

Table 29 – 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 41 and Figure 42 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the local queue manager scenario.

3.4.1.1 Non-persistent Messages

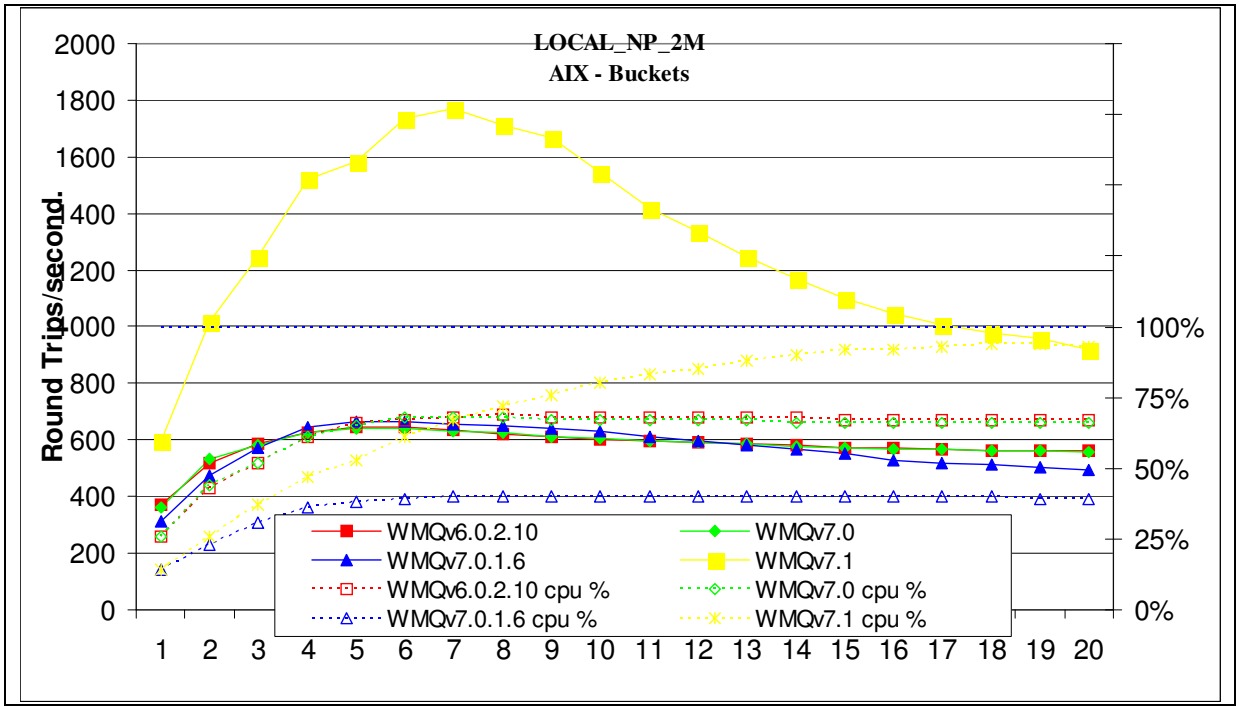


Figure 41 – 2MB non-persistent messages, local queue manager

Figure 41 and Table 30 shows that the throughput of non-persistent messages has increased by over 100% when comparing version 7.1 to V7.0.

Test Name: LOCAL_NP_2M	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	5	646	0.0084	66%
WMQv7.0	5	642	0.0084	65%
WMQv7.0.1.6	6	667	0.01	39%
WMQv7.1	7	1772	0.0044	67%

Table 30 – 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

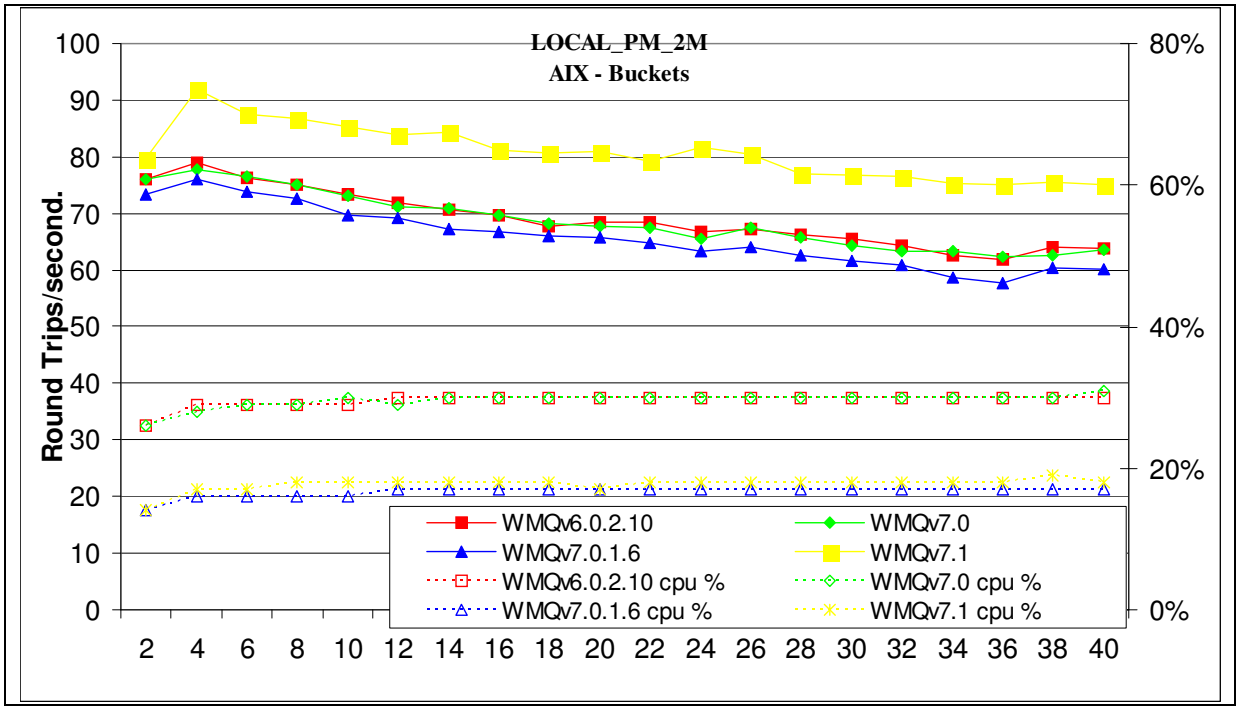


Figure 42 – 2MB persistent messages, local queue manager

Figure 42 and Table 31 shows that the throughput of persistent messages has increased by 18% when comparing version 7.1 to V7.0.

Test Name: LOCAL_PM_2M	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	4	79	0.054	29%
WMQv7.0	4	78	0.055	28%
WMQv7.0.1.6	4	76	0.057	16%
WMQv7.1	4	92	0.047	17%

Table 31 – 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 43 and Figure 44 show the non-persistent and persistent message throughput achieved using an increasing number of driving applications in the client channel scenario.

3.4.2.1 Non-persistent Messages

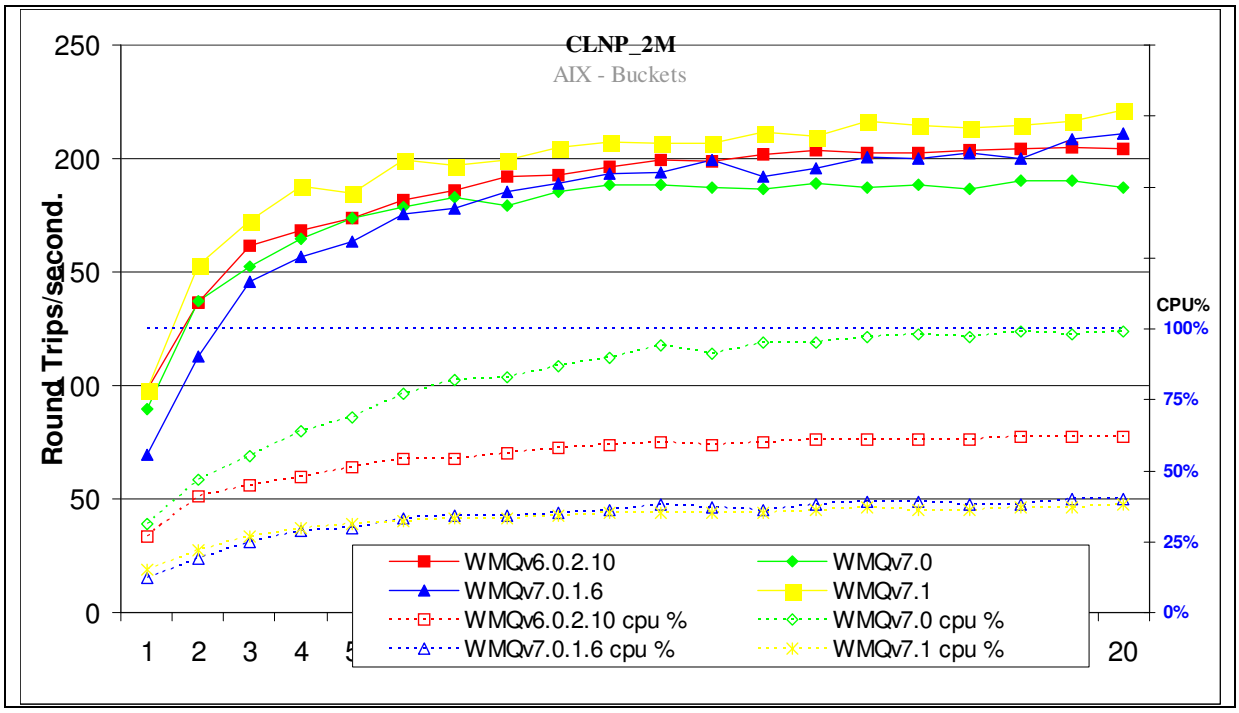


Figure 43 – 2MB non-persistent messages, client channels

Figure 43 and Table 32 shows that the throughput of non-persistent messages has increased by 12% when comparing version 7.1 to V7.0.

Test Name: CLNP_2M	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	19	205	0.13	62%
WMQv7.0	18	191	0.11	99%
WMQv7.0.1.6	20	211	0.16	40%
WMQv7.1	20	221	0.095	38%

Table 32 – 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

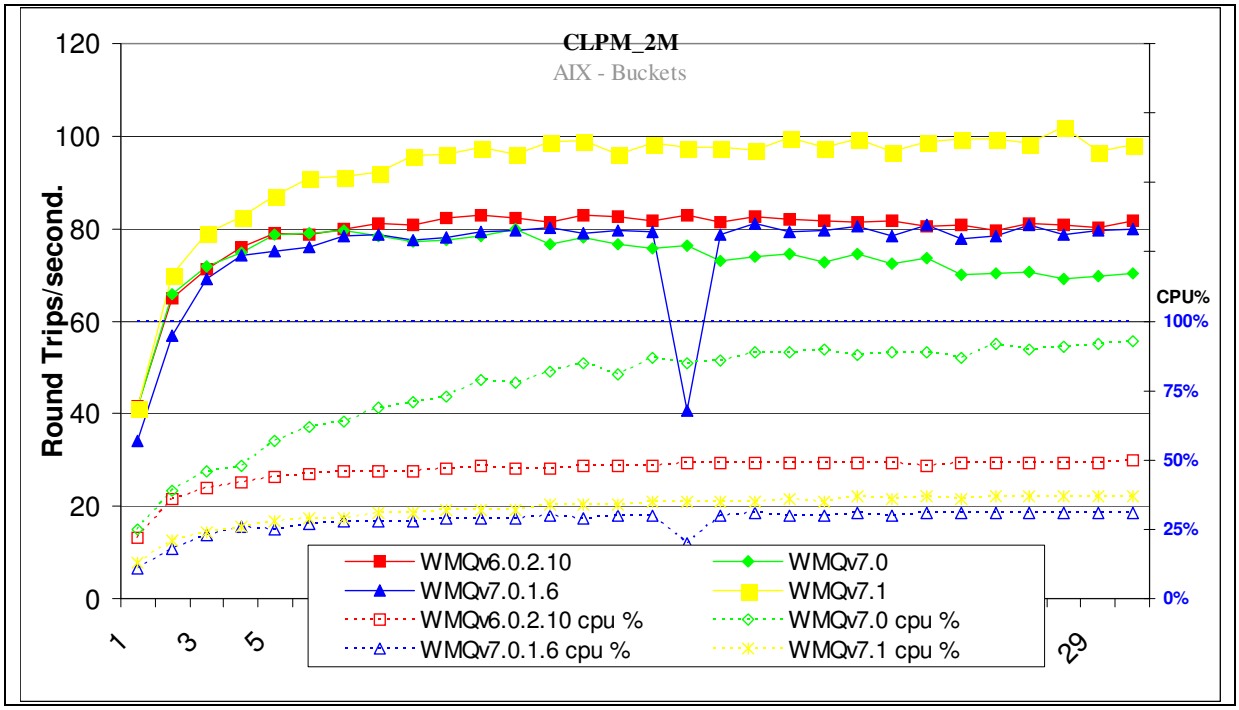


Figure 44 – 2MB persistent messages, client channels

Figure 44 and Table 33 shows that the throughput of persistent messages has increased by 27% when comparing version 7.1 to V7.0.

Test Name: CLPM_2M	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	14	83	0.21	48%
WMQv7.0	12	80	0.16	78%
WMQv7.0.1.6	19	81	0.27	31%
WMQv7.1	28	102	0.31	37%

Table 33 – 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 45 and Figure 46 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

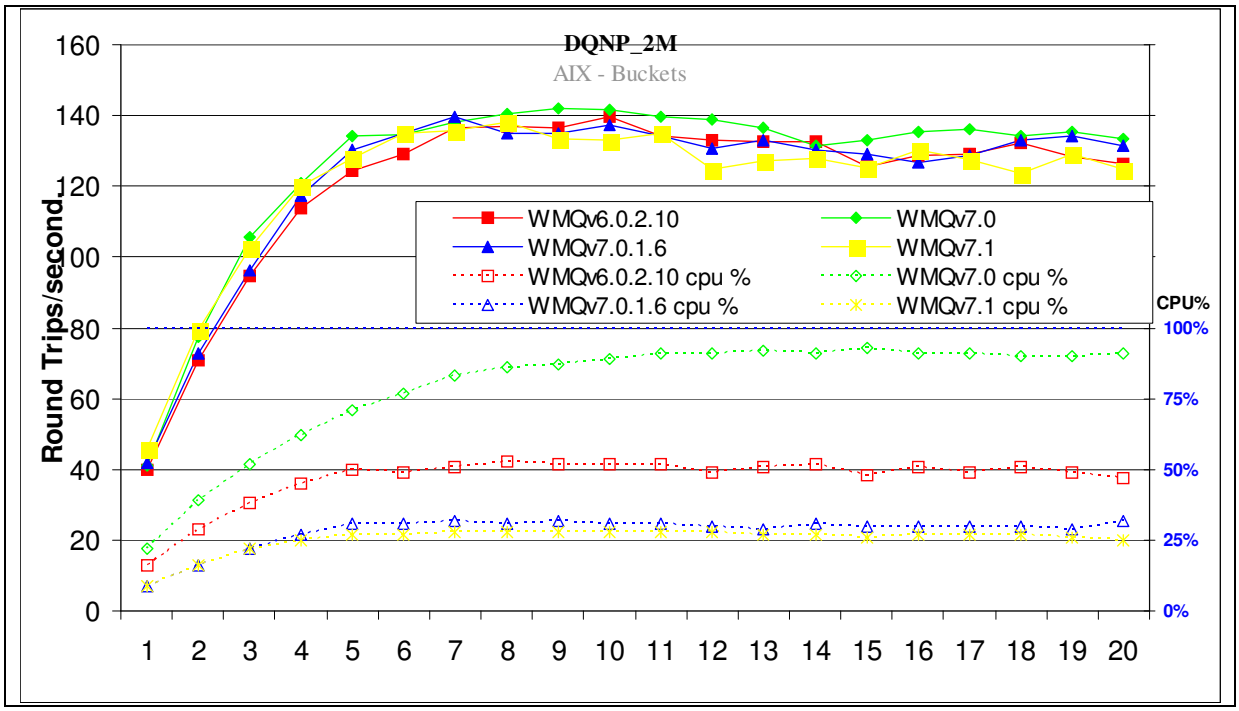


Figure 45 – 2MB non-persistent messages, distributed queuing

Figure 45 and Table 34 shows that the throughput of non-persistent messages has decreased by 4% when comparing version 7.1 to V7.0 but the CPU cost per messages has been halved.

Test Name: DQNP_2M	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	10	140	0.085	52%
WMQv7.0	9	142	0.076	87%
WMQv7.0.1.6	7	140	0.066	32%
WMQv7.1	8	138	0.072	28%

Table 34 – 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

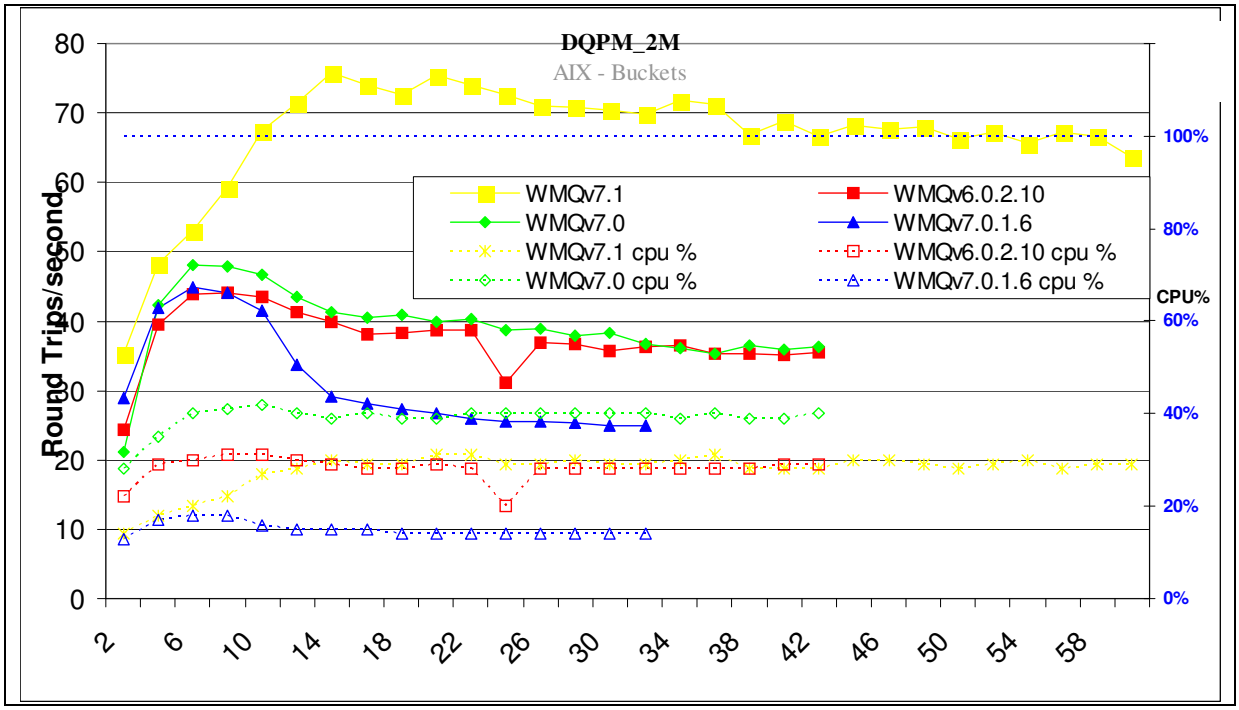


Figure 46 - 2MB persistent messages, distributed queuing

Figure 46 and Table 35 shows that the throughput of persistent messages has increased by over 100% when comparing version 7.1 to 7.0.

Test Name: DQPM_2M	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv6.0.2.10	8	44	0.2	31%
WMQv7.0	6	48	0.14	40%
WMQv7.0.1.6	6	45	0.15	18%
WMQv7.1	14	76	0.22	30%

Table 35 – 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 using V7.1.

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

4.1 Local Queue Manager

Figure 47 and **Figure 48** show the 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

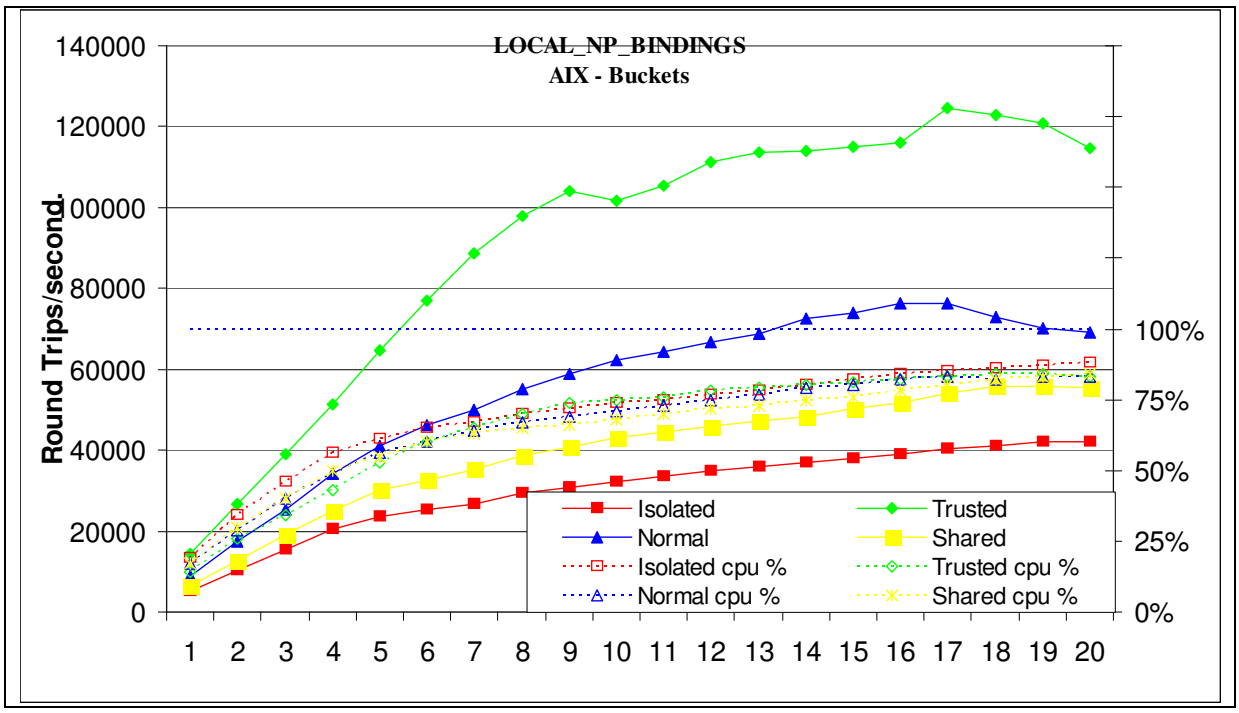


Figure 47 – Application binding, non-persistent messages, local queue manager

Figure 47 and Table 36 show that the throughput of non-persistent messages when comparing Normal, Isolated, Trusted and Shared bindings. The peak rate for Trusted bindings is over twice that for Shared binding and 195% faster than Isolated bindings.

Test Name: LOCAL_NP_BINDINGS	Apps	Round Trips/Sec	Response time (s)	CPU
Isolated	20	42207	0.0006	88%
Trusted	17	124696	0.00017	83%
Normal	17	76457	0.00027	83%
Shared	18	55900	0.00037	82%

Table 36 – Application binding, non-persistent messages, local queue manager

4.1.2 Persistent Messages

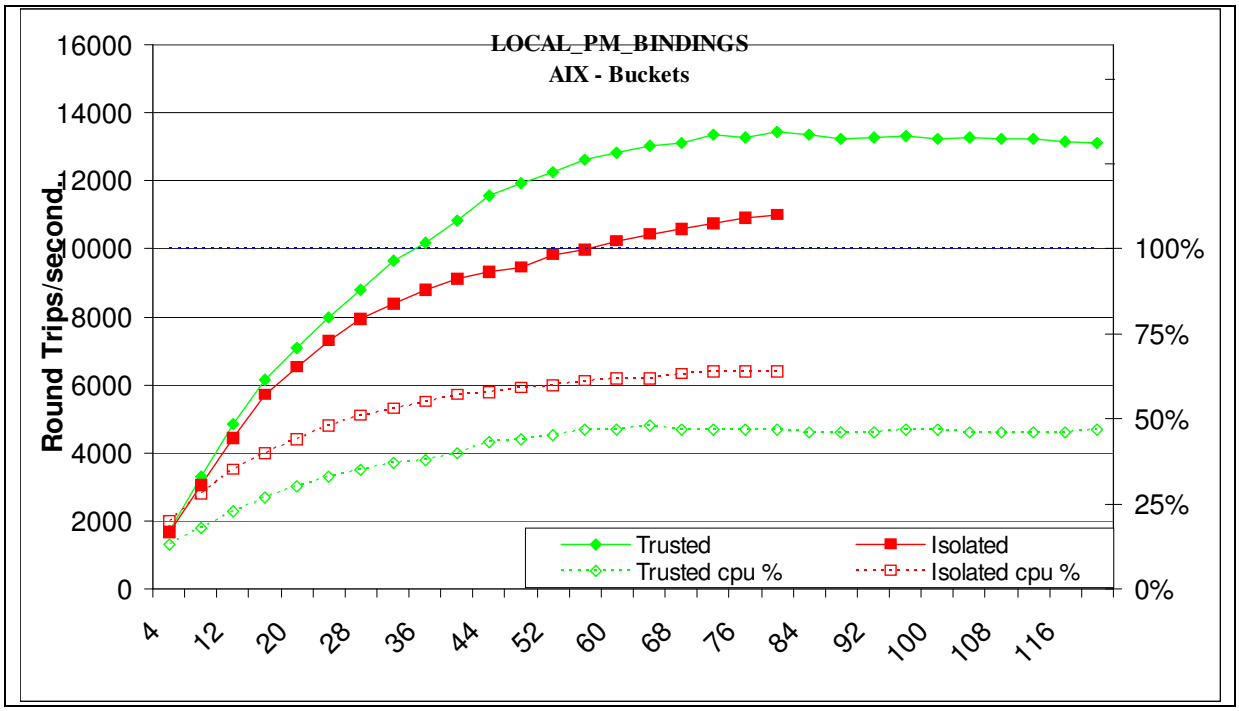


Figure 48 – Application binding, persistent messages, local queue manager

Figure 48 and Table 37 show that the throughput of persistent messages when comparing Isolated and Trusted bindings. Trusted bindings are 22% faster than Isolated.

Test Name: LOCAL_PM_BINDINGS	Apps	Round Trips/Sec	Response time (s)	CPU
Trusted	80	13438	0.0071	47%
Isolated	80	10988	0.0086	64%

Table 37 – Application binding, persistent messages, local queue manager

.

4.2 Client Channels

Figure 49 and **Figure 50** 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

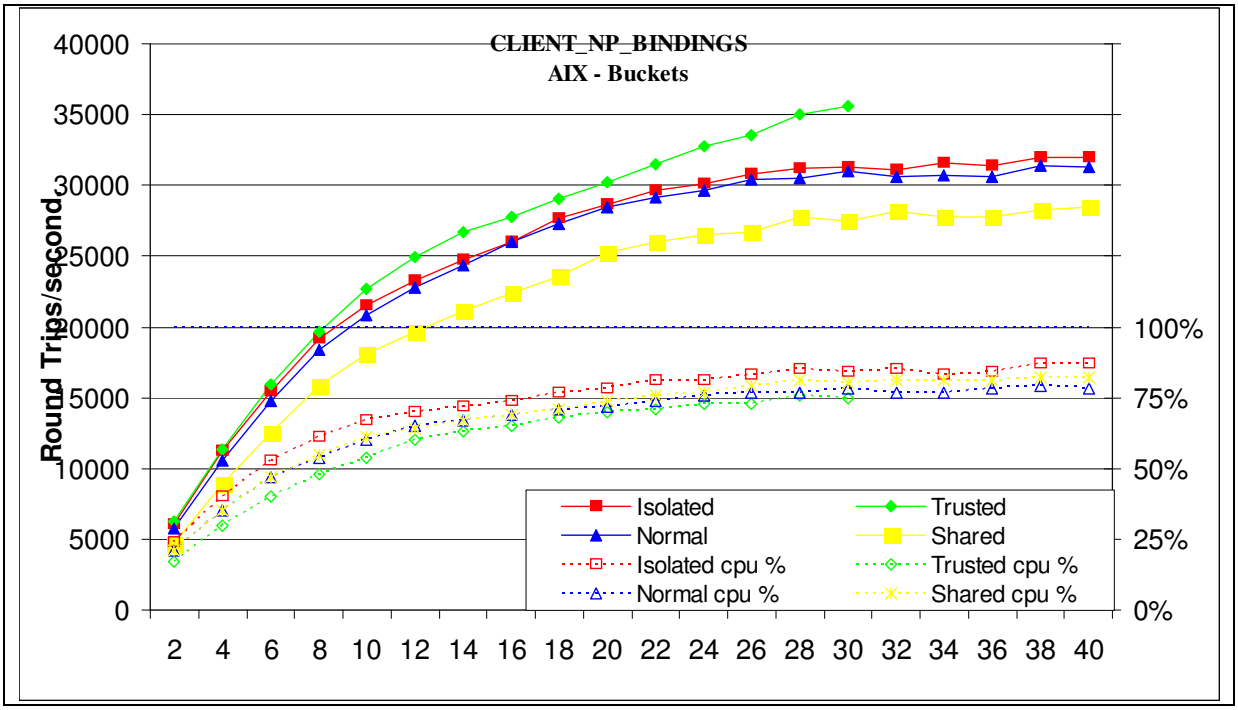


Figure 49 – Application binding, non-persistent messages, client channels

Figure 49 and Table 38 show that the peak throughput of non-persistent messages when comparing Normal, Isolated and Trusted bindings. Isolated bindings are 13% better than Shared bindings and Trusted bindings are 26% faster than Shared bindings

Test Name: CLIENT_NP_BINDINGS	Apps	Round Trips/Sec	Response time (s)	CPU
Isolated	40	31960	0.0014	87%
Trusted	30	35610	0.00097	75%
Normal	38	31383	0.0015	79%
Shared	40	28494	0.0016	82%

Table 38 – Application binding, non-persistent messages, client channels

4.2.2 Persistent Messages

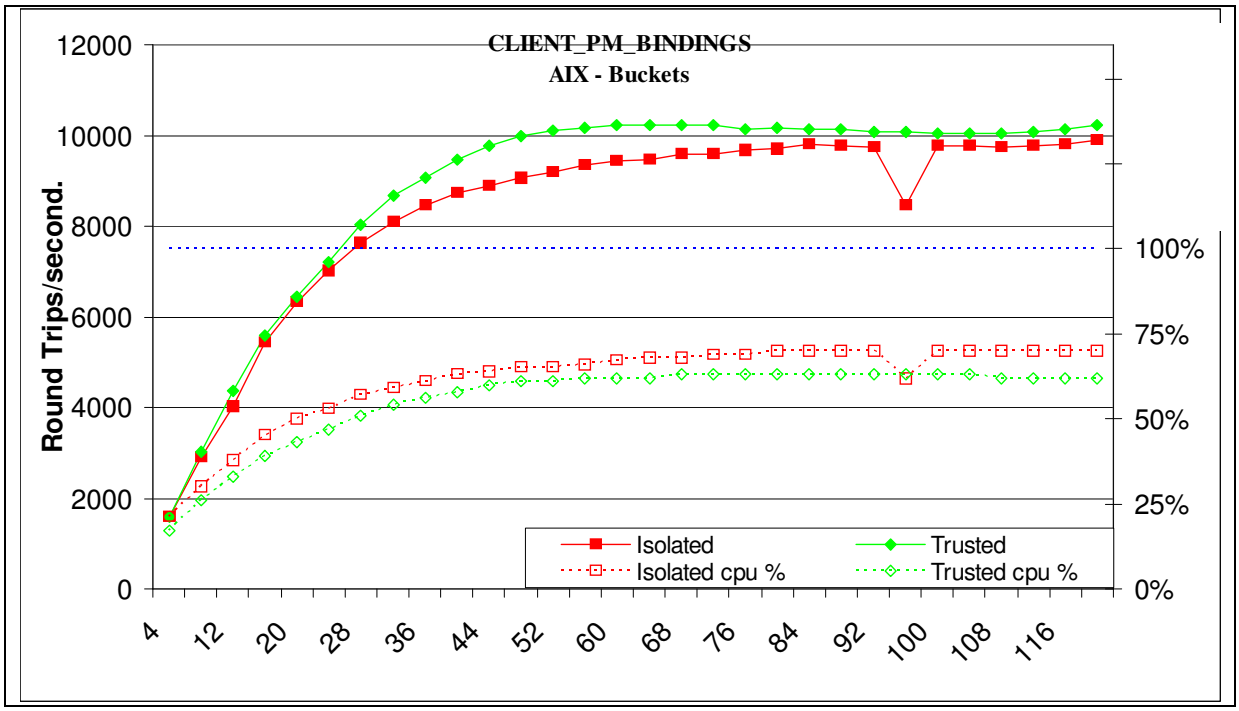


Figure 50 – Application binding, persistent messages, client channels

Figure 50 and Table 39 show that the peak throughput of non-persistent messages when comparing Isolated and Trusted bindings. The peak rate of Trusted bindings are 3% faster than Isolated

Test Name: CLIENT_PM_BINDINGS	Apps	Round Trips/Sec	Response time (s)	CPU
Isolated	120	9881	0.014	70%
Trusted	120	10231	0.013	62%

Table 39 – Application binding, persistent messages, client channels

.

4.3 Distributed Queuing

Figure 50 and **Figure 51** 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

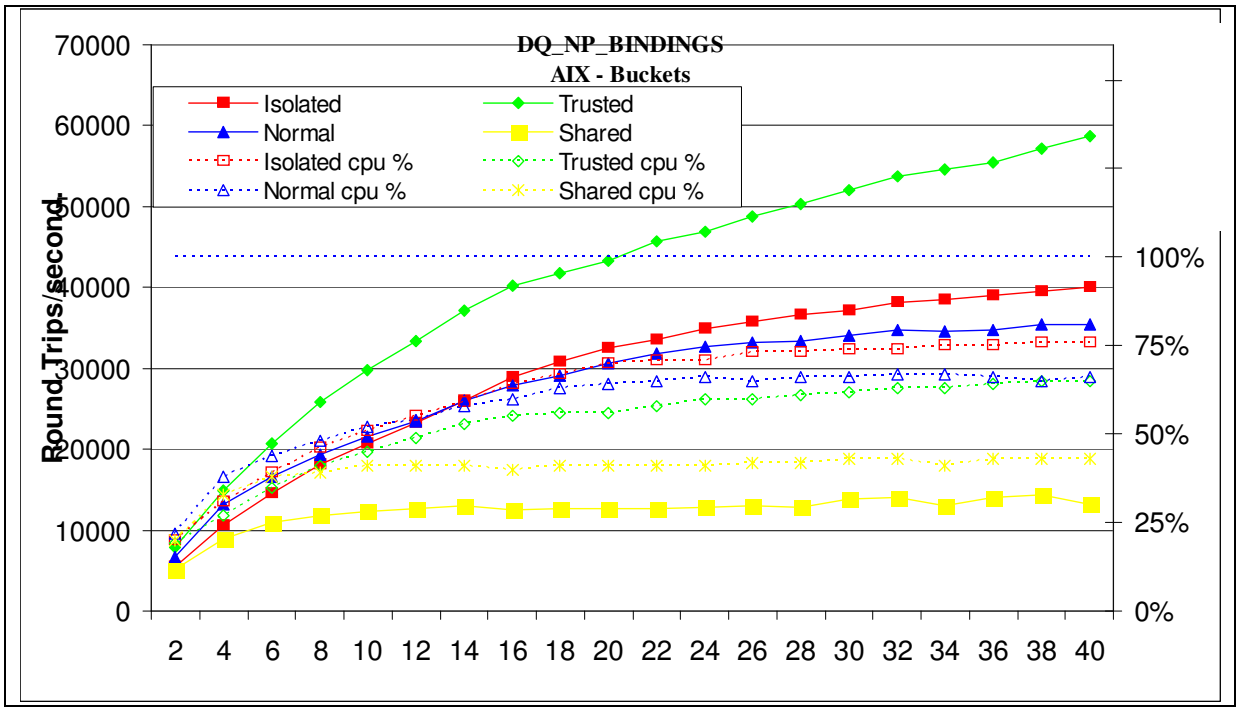


Figure 51 – Application binding, non-persistent messages, distributed queuing

Figure 51 and Table 40 show that the peak throughput of non-persistent messages when comparing Normal, Isolated and Trusted bindings. The peak rate for Trusted bindings is over 4 times the rate for Shared while Isolated is 3 times the rate for Shared bindings.

Test Name: DQ_NP_BINDINGS	Apps	Round Trips/Sec	Response time (s)	CPU
Isolated	40	40126	0.0011	76%
Trusted	40	58703	0.00077	65%
Normal	40	35492	0.0013	66%
Shared	38	14435	0.0036	43%

Table 40 – 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

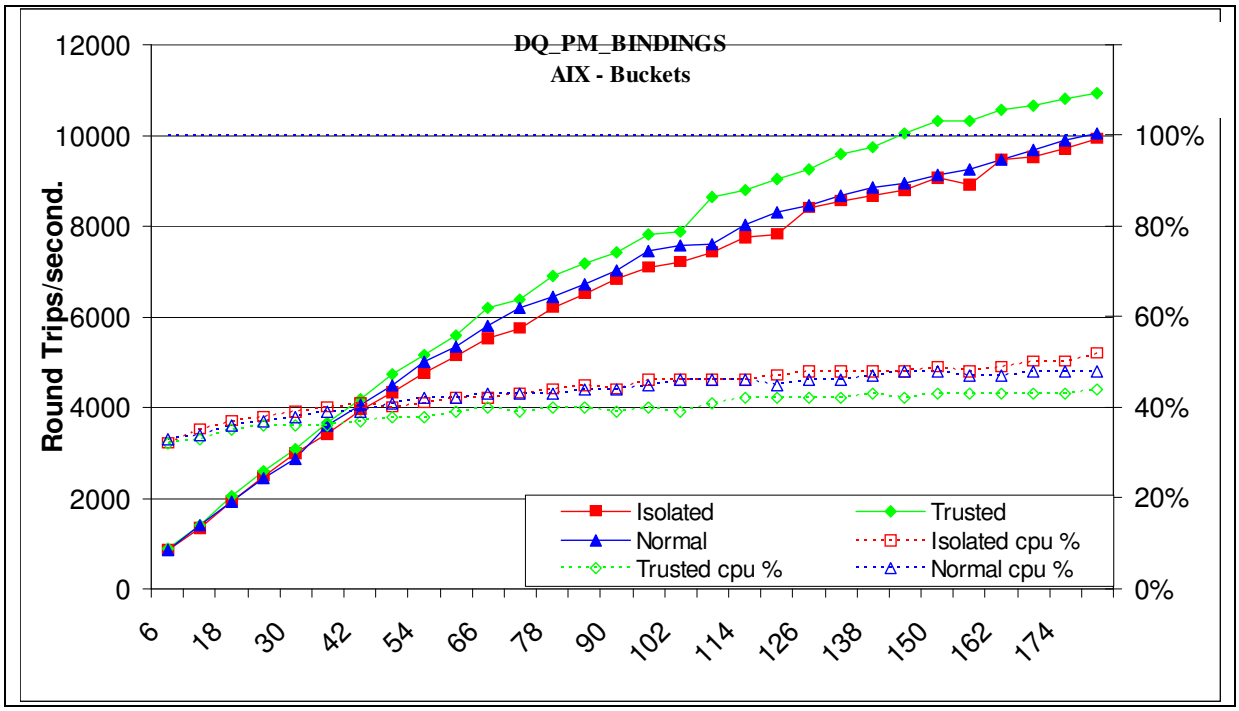


Figure 52 – Application binding, persistent messages, distributed queuing

Figure 52 and Table 41 show that the peak throughput of non-persistent messages when comparing Isolated and Trusted bindings. The peak rate for Trusted bindings is 10% faster than for Isolated bindings.

Test Name: DQ_PM_BINDINGS	Apps	Round Trips/Sec	Response time (s)	CPU
Isolated	180	9913	0.02	52%
Trusted	180	10940	0.017	44%
Normal	180	10048	0.022	48%

Table 41 – Application binding, persistent messages, distributed queuing

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

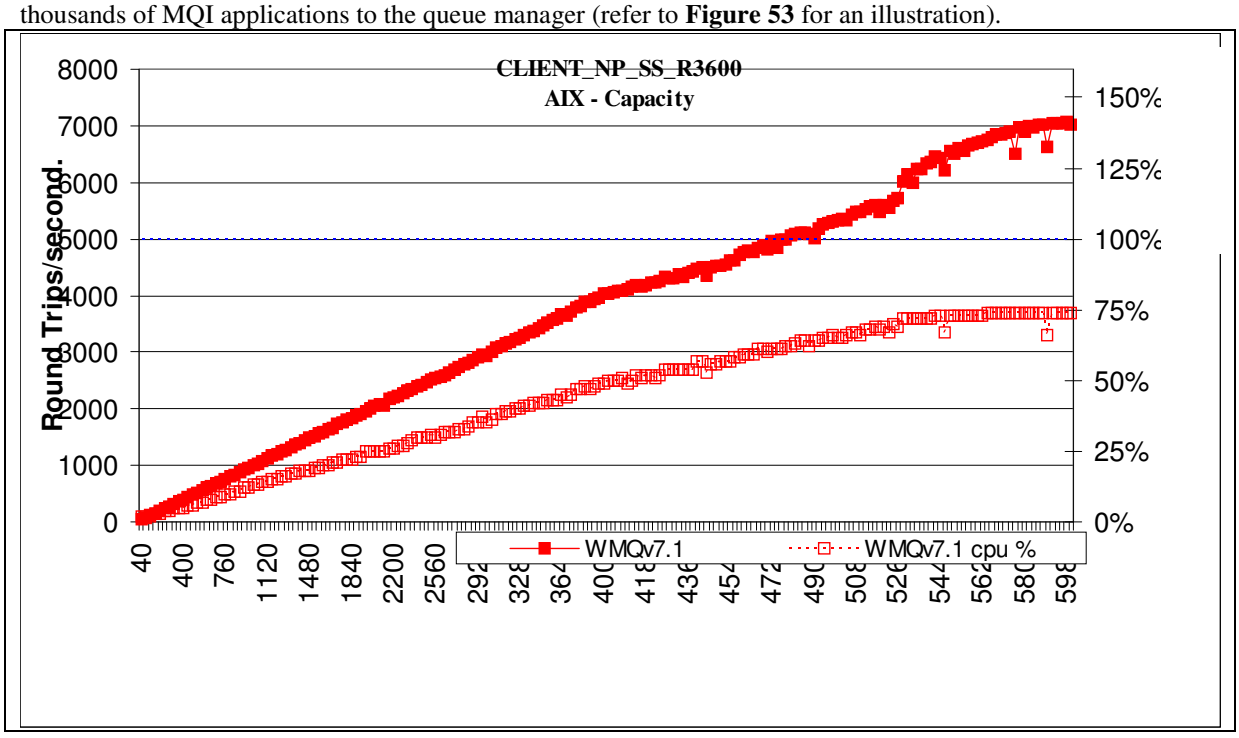
5 Short & Long Sessions

The previous chapters in this report only reported on steady state messaging that does not include any session setup and termination function. This chapter specifically bracket groups of five MQPut/MQGet pairs with MQConn/MQDisc and MQOpen/MQClose calls so a comparison of this overhead can be seen.
A short session is a term used to describe the behaviour of an MQI application as it processes a small number of messages using one or more queues and a queue manager. The measurements in this document use an MQI-client application and the following sequence:

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

"Why measure short sessions?"

For each new connecting application or disconnecting application, the queue manager and Operating System must start a new process or thread and set up the new connection. As the number of connecting and disconnecting applications increases, the Operating System and queue manager are subjected to a higher load. While these requests are being serviced, the queue manager has less time available to process messages, so fewer driving applications can be reconnected to the queue manager per second before the response time exceeds one second.
This effect is greater than that of reducing the total messaging throughput of the queue manager by connecting thousands of MQI applications to the queue manager (refer to **Figure 53** for an illustration).



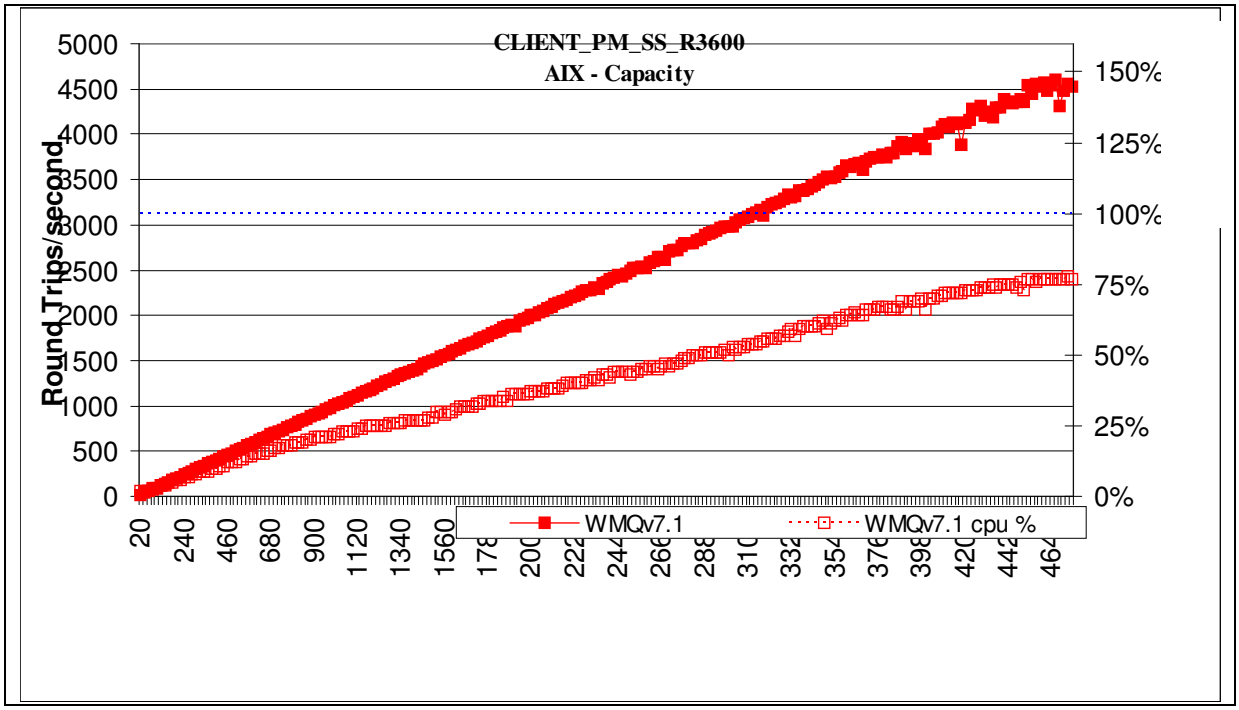


Figure 53 – Short sessions, client channels

Test Name: CLIENT_NP_SS_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv7.1	5860	7024	0.97	74%

Test Name: CLIENT_PM_SS_R3600	Apps	Round Trips/Sec	Response time (s)	CPU
WMQv7.1	4560	4555	0.74	76%

Table 42 – 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.

5860 round trips is 1172 short sessions per second and 4560 round trips is 912 short sessions.

These short session results required APAR IV10239.

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	38	n/a*	31363	0.0015	79%
clnp_r3600	13900	3600	13898	0.0014	53%
clnp_c6000	6000	13260	22094	0.008	76%
clnp_cmax	50000	390 570 630	5515 8051 8728	0.0098 0.044 0.697	35% 45% 50%
cl_persist_c6000	6000	950 2100 3580	1572 3536 5857	0.034 0.34 0.99	23% 40% 55%
clnp_c6000_no_correllid	6000	9900	16497	0.009	79%

Table 43 – 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 whose memory could be defined by using the RMSS facility. 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
4GB	27400	13000	12600
5GB	35800	17800	17700
6GB	43900	22900	22400

Linear approximations of these points enable the amount of working set 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	672MB	1383MB	1410MB
Additional clients	121K	202K	204K

On Version 7.1, a measurement was made with each client having its own dynamic queue and the result was that 10900 clients could be successfully attached. Hence each Queue uses 32K of working set storage.

6.1.2 Client channels – memory migration cost

Costs associated with customer migration path 6.0.2.11 to 7.1

I. OS + QMGR costs increased from 672 MB to 1410 MB
II. Cost of a client has increased from 121 KB to 204 KB

Costs associated with customer migration path 7.0.1.6 to 7.1

I. OS + QMGR costs increased from 1383 MB to 1410 MB

Cost of a client has increased from 202 KB to 204 KB

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	40	n/a*	35492	0.016	66%
dqnp_r3600	19700	3600	19698	0.00081	48%
dqnp_q1000	1000	60000	16616	0.0007	45%
dqnp_q7000	7000	7200	13997	0.002	65%
dq-persist_q1000	1000	13000 22900	3607 5748	0.009 0.087	40% 55%
dq_persist_q4000	4000	6100 11100	3638 6170	0.309 0.572	41% 67%

Table 44 – 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, Oracle 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 (AIX)

These settings are made to enable the system to cope with the full set of MQ Benchmarks
maxuproc=64000
no -r -o tcp_ephemeral_low=1024
AIXTHREAD_SCOPE=S command to the **/etc/environment** file.
etc/security/limits

```
mqm:
        fsize = 2097151
        core = 2097151
        cpu = -1
         data = 262144
         rss = 65536
        stack = 65536
        nofiles = 100000
```

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 additional 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

There are two common methods used to compare throughput graphs namely the peak messaging rate or the area under the graph. In this document, the percentage throughput comparison used the area under the graph as the method of interpreting the performance data. Other MQ Performance reports use the percentage throughput comparison using 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

IBM Power 6 :	Server system (Device under test)
Model:	IBM,9117-MMA P570
Processor:	4.2GHz
Architecture:	8 core
Memory (RAM):	16GB
Disk:	2 Internal 16bit SCSI (9GB each, 1 O/S, swap)
	2 SAN disks on DS8700 (5GB each, 1 queue, 1 log)
Network:	10Gbit Ethernet Adapter
IBM power 7:	Driver system
Model:	IBM,8233-E8B P750
Processor:	3.55GHz
Architecture:	8 core
Memory (RAM):	32GB
Disk:	2 Internal 16bit SCSI (9GB each, 1 O/S, swap)
	2 SAN disks on DS8700 (5GBb 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

AIX:	AIX **7100-00 SP3 (plus IZ87155 IV09133, IV09134, IV09144, IV06100)**
MQSeries:	Version 6.0.1.10, Version 7.0, Version 7.0.1.6 Version 7.1
Compiler:	C for AIX Compiler, Version 7.0

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