WebSphere MQ for HP-UX V5.3 - Performance Evaluations

Version 1.1

5th August 2002

Alexander Russell. M.Eng. IBM Certified

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

Take Note!

Before using this report, be sure to read the general information under “Notices”.

First Edition, July 2002
Second Edition, August 2002
This edition applies to V1.0 of WebSphere MQ for HP-UX V5.3 – Performance Evaluations and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

This report is intended to help the reader understand the performance characteristics of WebSphere MQ for HP-UX V5.3. The information is not intended as the specification of any programming interfaces that are provided by WebSphere MQ.

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which it operates.

Information contained in this report has **not** been submitted to any formal IBM test and is distributed "**as-is**". The use of this information and the implementation of any of the techniques is the responsibility of the customer. Much depends on the ability of the reader to evaluate the information and project the results to their operational environment.

The performance measurements included in this report were measured in a controlled environment and the results obtained in other environments may vary significantly.

Trademarks and service marks:

Preface

Target audience

This SupportPac is designed for people who:

- Will be designing and implementing solutions using WebSphere MQ for HP-UX
- Want to understand the performance limits of WebSphere MQ for HP-UX V5.3
- Want to understand what actions may be taken to tune WebSphere MQ for HP-UX

The reader should have a general awareness of the HP-UX 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 V** 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 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 manner more useful to you? Please direct any comments of this nature to: **WMQPG@uk.ibm.com**.

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

Acknowledgements

The author is very grateful to Richard Eures for help in producing this report.

Introduction

The three scenarios in this report used to generate the performance data are classified into: the local queue manager scenario, the client channel scenario, and the distributed queuing scenario. The performance improvements in WebSphere MQ V5.3 can be divided into two areas:

- queue manager enhancements, and
- channel capacity enhancements.

The enhancements to the queue manager are apparent through many of the measurements in this report where WebSphere MQ V5.3 is compared to Version 5.2. Channel capacity enhancements are covered briefly in the *release highlights* section and in more detail towards the end of the report.

Unless otherwise specified, the standard message sized used for all the measurements in this report is 2K (2,048 bytes), trusted channels using the inetd ‘amqcrsta’ listener, and nontrusted threaded server application are used.

An HP-UX (model 9000/800/L2000-44) 4-way SMP 440MHz with 4GB of RAM was used as the device under test for all the measurements in this report.

How this document is arranged

Release highlights

Pages: 1-4

Section one outlines the major performance *improvements* achieved in WebSphere MQ V5.3 compared to Version 5.2. The highlights are a subset of the results shown in the performance *headlines* section.

Performance headlines

Pages: 5-20

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

- a local queue manager,
- to a remote queue manager over MQI-client channels, and
- to 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, and
- the *maximum* number of server channel pairs between two queue managers, for a fixed message rate until the response time exceeds one second.

Large messages

Pages: 22-31

Section three of the document contains performance measurements for *large messages*. This includes *MQI response times* of 50byte to 2MB messages, and *20K* and *200K* messages using the same test scenarios as for the performance *headlines*.

Trusted server application

Pages: 35

Section four contains performance measurements for a *trusted* server application, using the three test scenarios as for the performance *headlines*.

Short sessions

Pages: 37

Section five contains performance measurements for *short sessions*. That is, an MQI application connecting to the queue manager, processing a few messages between connecting to and disconnecting from the queue manager.

Capacity measurements

Pages: 39-41

Section six of this document shows:

- the total number of MQI-client channels that were connected into a single queue manager, with a server application processing 1 nonpersistent round trip per MQI-client per *minute*.
- the total number of server channel pairs that were connected between two queue managers on separate server machines, with a server application processing 1 nonpersistent round trip per server channel pair per *minute*.

Tuning recommendations:

Pages: 43—

In previous SupportPacs tuning recommendations have been in a separate section. In this document queue manager parameters are mentioned with the measurements they are appropriate to, with detailed discussion in '**Tuning recommendations**' starting on **page 43**.

Measurement environment:

Pages: 48-49

A summary of the way in which the workload is used in each test scenario is given in the performance headlines section. For a more detailed description of the workload, hardware and software specifications, refer to the '**Measurement environment**'—**Page 48**.

Glossary:

Pages: 50

A short glossary of the terms used in the tables throughout this document can be found in the '**Glossary**'—**Page 50**.

CONTENTS

TABLES

FIGURES

1 Release highlights

1.1 Improvements to nonpersistent and persistent messaging

- Nonpersistent message throughput increased by an average of: 15% in a local queue manager environment, 5% in an MQI-client environment, and 7% in a distributed queuing environment.
- Persistent message throughput increased by an average of: 31% in a local queue manager environment, 28% in an MQI-client environment, and 8% in a distributed queuing environment.

1.2 Peak message throughput – local queue manager

Figure 1 below shows the peak round trips per second achieved for nonpersistent and persistent messages with a local queue manager, MQSeries V5.2 vs. WebSphere MQ V5.3.

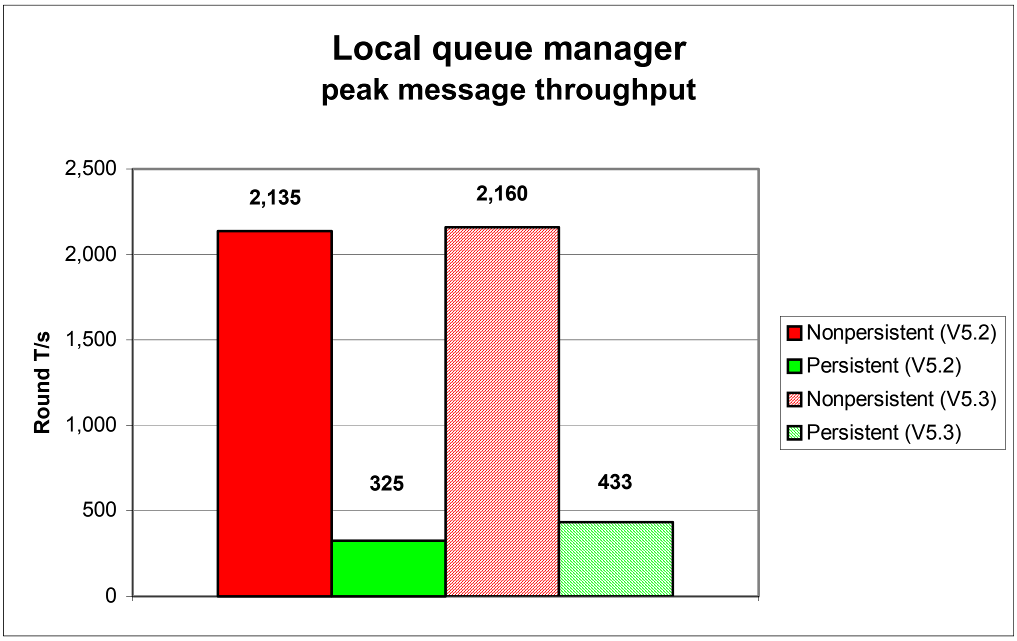

Figure 1 – Peak message throughput, local queue manager

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

1.3 Peak message throughput – client channels

Figure 2 below shows the peak round trips per second achieved for nonpersistent and persistent messages with MQI-client channels, MQSeries V5.2 vs. WebSphere MQ V5.3.

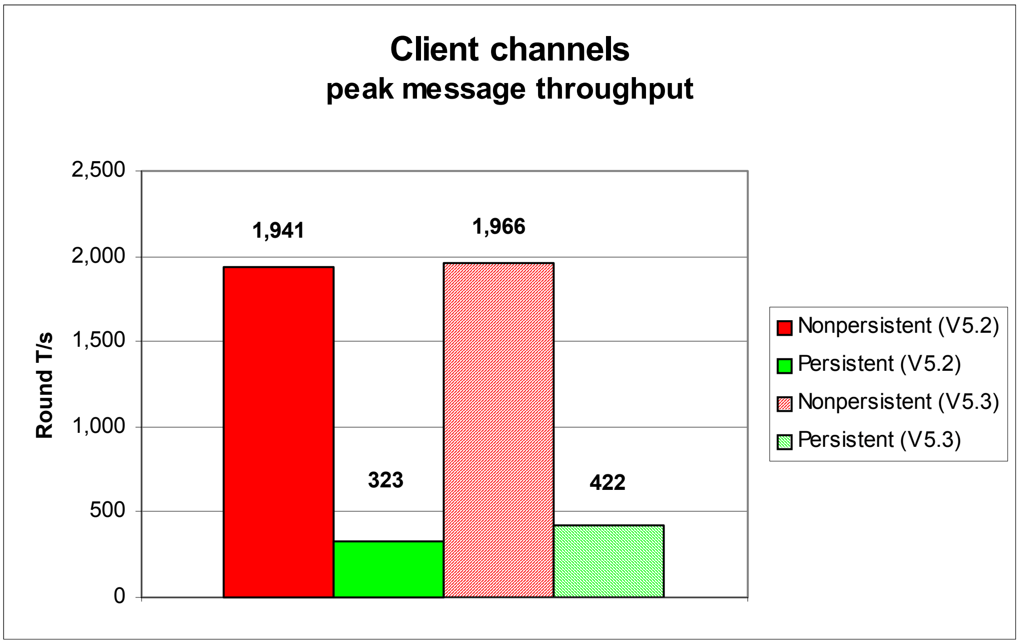

Figure 2 – Peak message throughput, client channels

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

1.4 Peak message throughput – distributed queuing

Figure 3 below shows the peak round trips per second achieved for nonpersistent and persistent messages with server channels, MQSeries V5.2 vs. WebSphere MQ for V5.3.

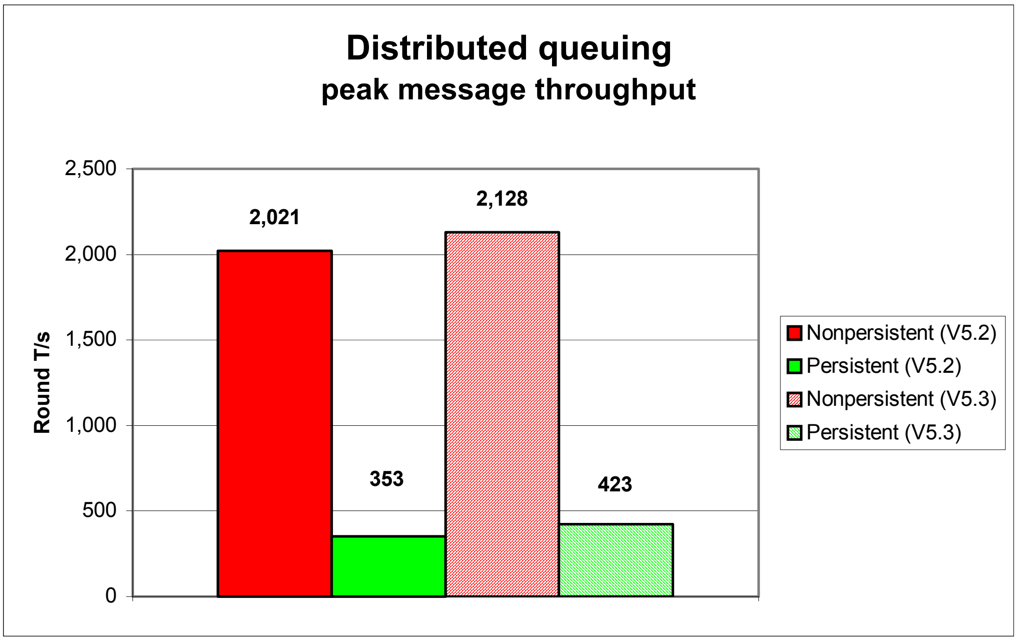

Figure 3 – Peak message throughput, distributed queuing

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

1.5 Improvements to channel capacity limits

- Client channels (trusted MQI-client connections) increased from 5,400 to 17,500 using one reply queue for all clients, and can support 7,500 using one reply queue per client.
- Distributed queuing (trusted server channels) increased from 2,340 to 5,000 pairs.

1.6 Capacity limits – client channels

Figure 4 below shows the maximum number of MQI-client connections made concurrently into a single queue manager on the server machine.

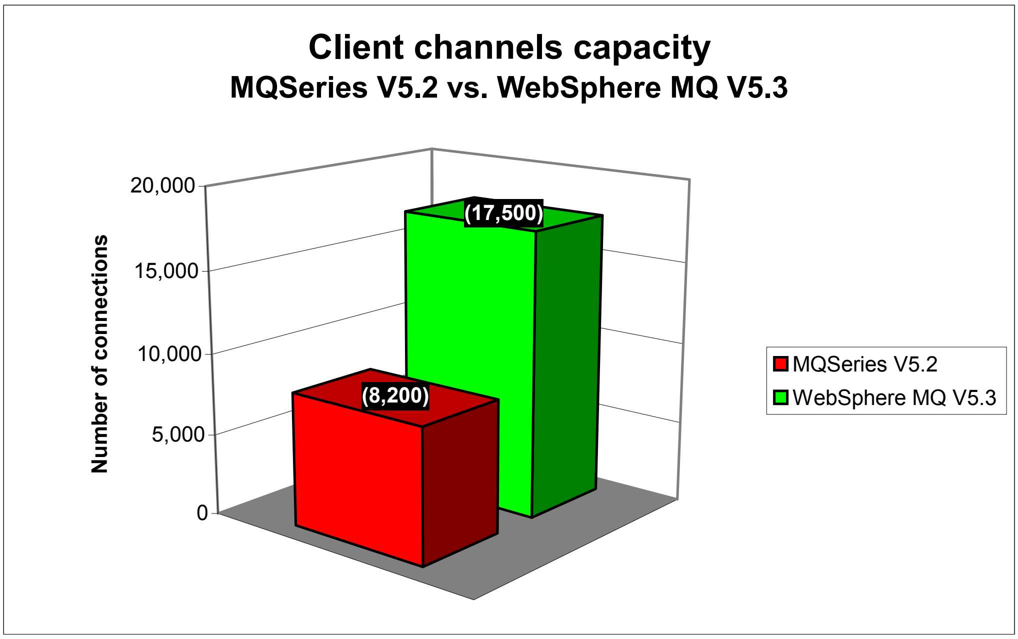

Figure 4 – Maximum number of client connections

Note: messaging in these tests uses a rate of 1 round trip per MQI-client per *minute*.

1.7 Capacity limits – distributed queuing (server channels)

Figure 5 below shows the maximum number of server channel pairs achieved between two queue managers on separate server machines.

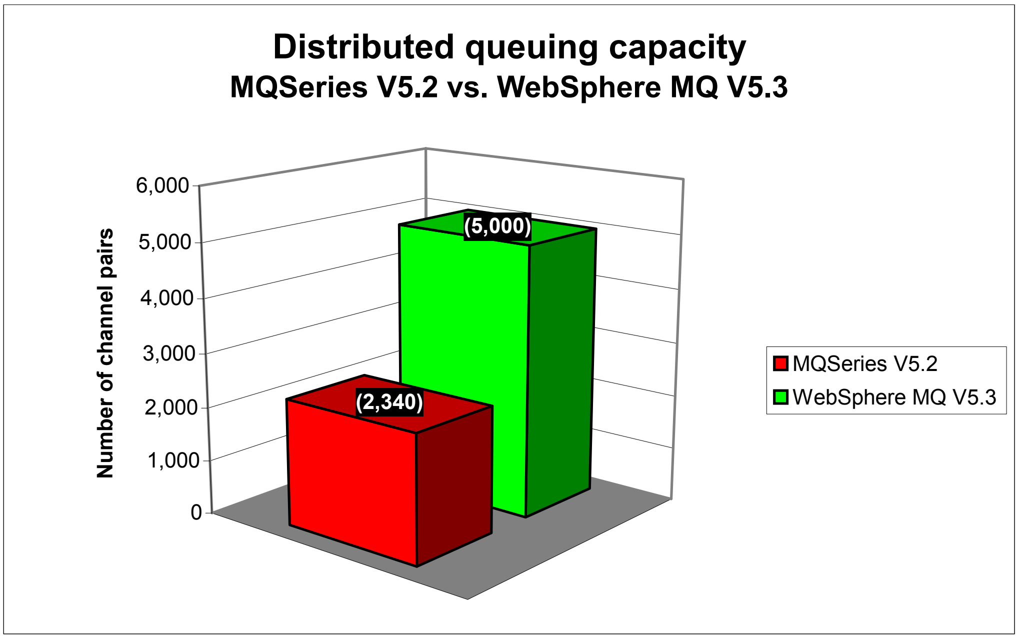

Figure 5 – Maximum number of server channels

Note: messaging in these tests uses a rate of 1round trip per server channel pair per *minute*.

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 defined as 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*.

In the rated messaging tests, the rate used is 1 round trip per driving application per *second*. In the client channel test scenarios, each driving application using a dedicated MQI-client channel, in the distributed queuing test scenarios, one or more applications submit messages over a fixed number of server channels.

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

2.1 Local queue manager test scenario

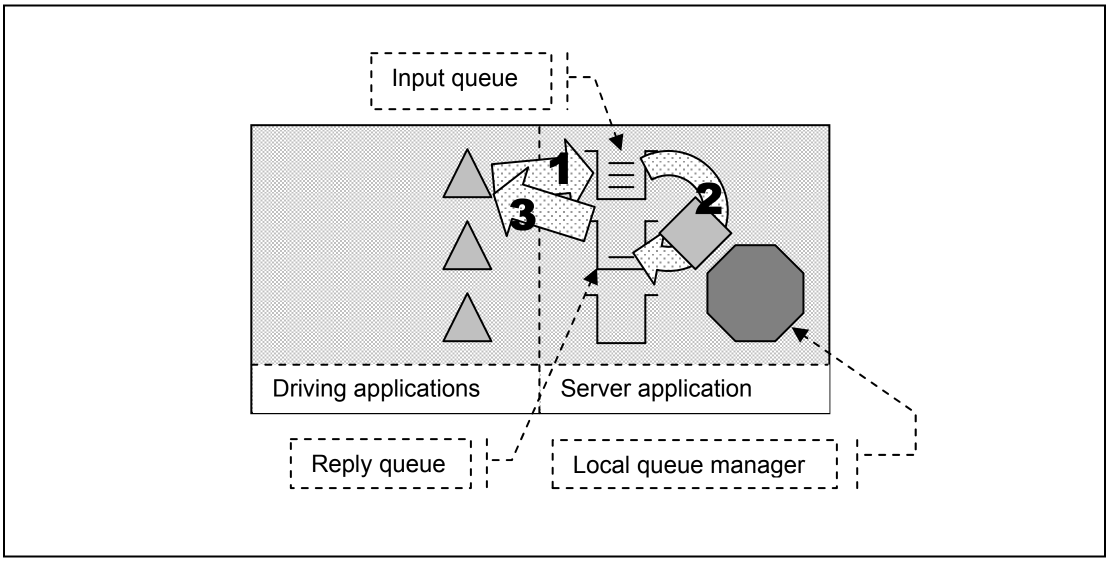

Figure 6 – Connections into a local queue manager

1) The driving 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 driving application then waits indefinitely for a reply to arrive on the common reply queue.

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

Nonpersistent and persistent messages were used in the local queue manager tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in '**MQI response times: 50bytes to 2MB – local queue manager**'—**Page 22**, and '**Large messages: 20K and 200K – local queue manager**'—**Page 25**.

Figure 7 and **Figure 8** below shows the nonpersistent and persistent message throughput achieved using an *increasing* number of driving applications in the local queue manager test scenario (see **Figure 6** above), and WebSphere MQ V5.3 compared to Version 5.2.

2.1.1 Nonpersistent messages – local queue manager

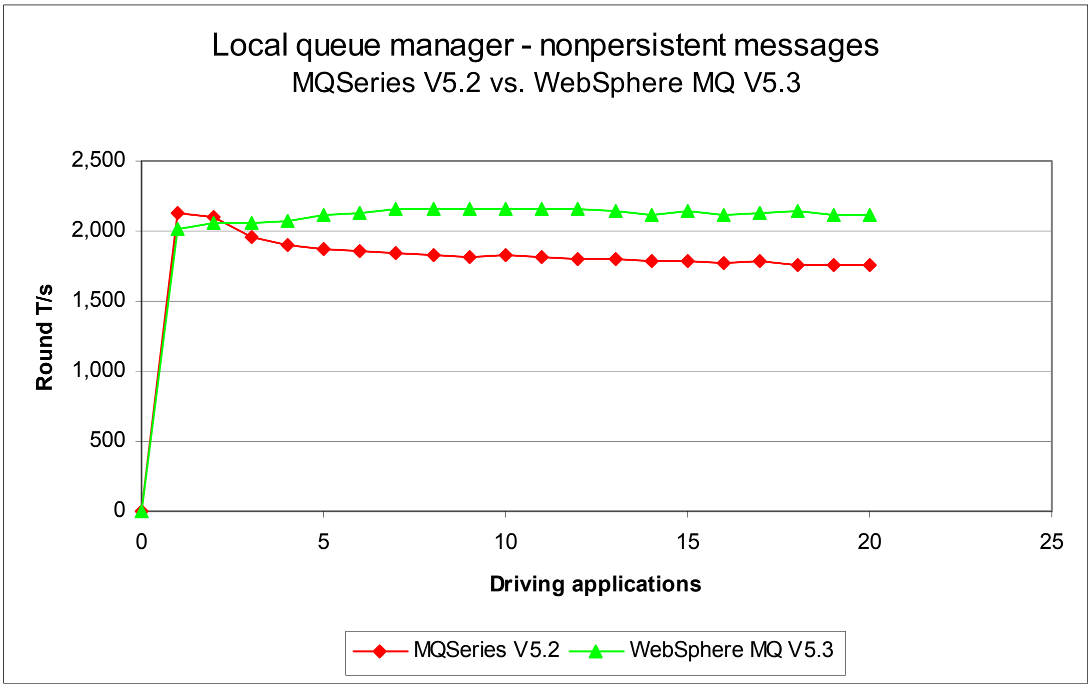

Figure 7 – Performance headline, nonpersistent messages, local queue manager

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

Test name	Product version	Apps	Round T/s	%	Resp time (s)
local_np1	MQSeries V5.2	**1** (8) (20)	**2,135** (1,830) (1,769)	n/a	0.001 (0.005) (0.013)
local_np1	WebSphere MQ V5.3	(1) **8** (20)	(2,016) **2,160** (2,139)	(-6) +18 (+21)	(0.001) 0.003 (0.011)

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

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

Figure 7 and **Table 1** above show that the peak throughput of nonpersistent messages has increased in Version 5.3 compared to Version 5.2, with the advantage of improved scalability when using as less then 5 driving applications. Using 8 driving applications nonpersistent throughput is improved by 18% (1,830 cf. 2,160 RT/s). Although Version 5.2 shows higher throughput for one or two driving applications, with three or more driving applications Version 5.3 shows a consistent throughput improvement—which is more important in most queue manager systems.

2.1.2 Persistent messages – local queue manager

Queue manager log configuration:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

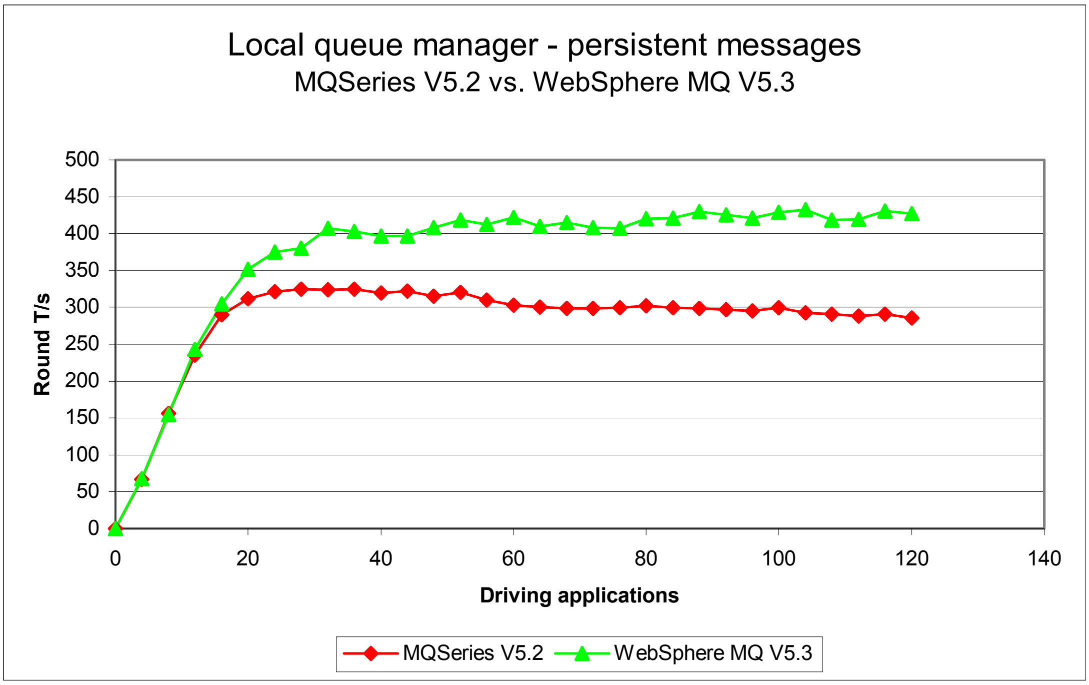

Figure 8 – Performance headline, persistent messages, local queue manager

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

Test name	Product version	Apps	Round T/s	%	Resp time (s)
local_pm1	MQSeries V5.2	**28** (104) (120)	**325** (293) (286)	n/a	0.107 (0.372) (0.429)
local_pm1	WebSphere MQ V5.3	(28) **104** (120)	(380) **433** (428)	(+17) +48 (+50)	(0.082) 0.288 (0.353)

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

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

Figure 8 and **Table 2** above show that the peak throughput of persistent messages has increased by 33% (325 cf. 433 RT/s) comparing Version 5.2 to Version 5.3. Using 28 driving applications persistent throughput is improved by 17% (325 cf. 380 RT/s). Version 5.3 has the advantage of improved scalability when using 20 or more driving applications.

2.2 Client channels test scenario

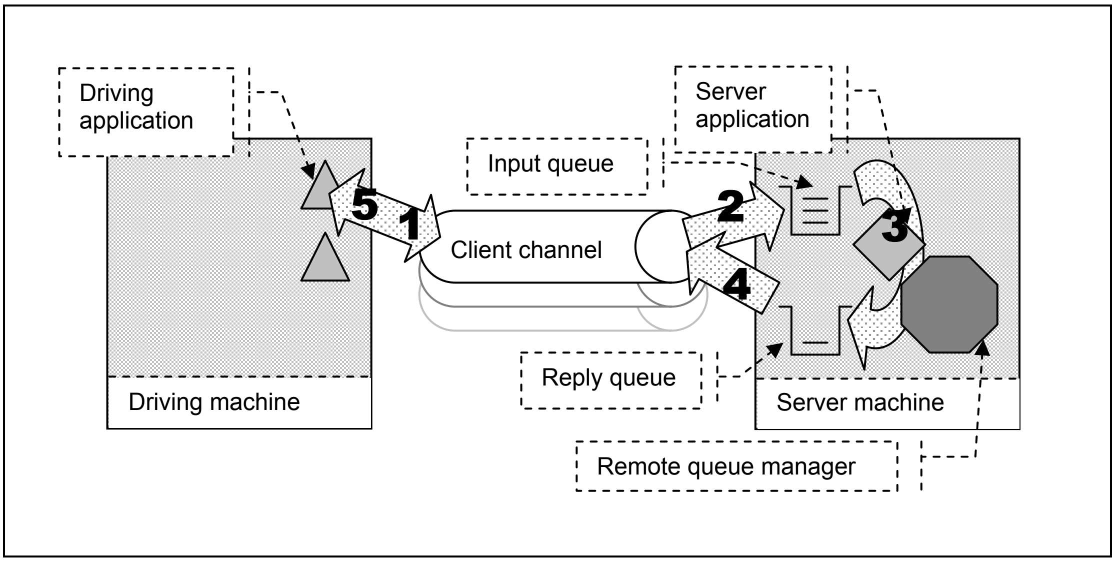

Figure 9 – MQI-client channels into a remote queue manager

1, 2) The driving 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 driving application then waits indefinitely for a reply to arrive on the common reply queue.

3) The server 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 driving application gets the reply message (over the client channel), from the common reply queue. The driving application uses the message identifier held from when the request message was put to the common input queue, as the correlation identifier in the message descriptor.

Nonpersistent and persistent messages were used in the client channel tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in '**Large messages: 20K and 200K – client channels**'—**Page 28**.

Figure 10 and **Figure 11** below shows the nonpersistent and persistent message throughput achieved using an *increasing* number of driving applications in the client channel test scenario (see **Figure 9** above), and Version 5.2 compared with Version 5.3.

2.2.1 Nonpersistent messages – client channels

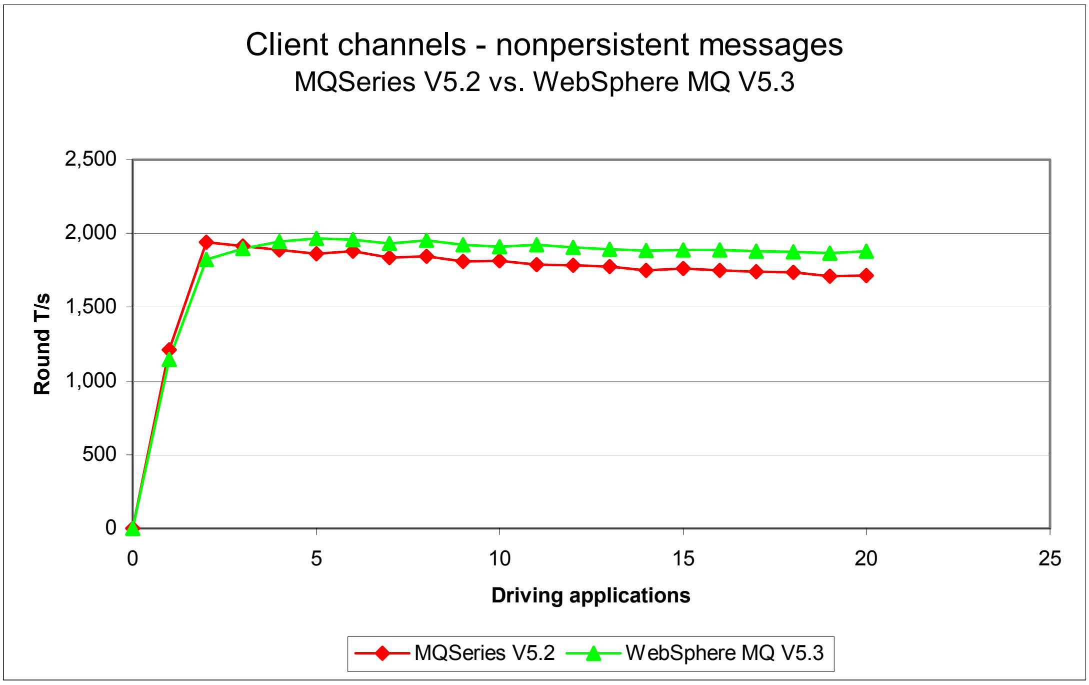

Figure 10 – Performance headline, nonpersistent messages, client channels

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

Test name	Product version	Apps	Round T/s	%	Resp time (s)
clnp1 (inetd)	MQSeries V5.2	**2** (5) (20)	**1,941** (1,863) (1,713)	n/a	0.001 (0.003) (0.014)
clnp1 (inetd)	WebSphere MQ V5.3	(2) **5** (20)	(1,823) **1,966** (1,879)	(-6) +6 (+10)	(0.001) 0.002 (0.012)

Table 3 – Performance headline, nonpersistent messages, client channels

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

Figure 10 and **Table 3** above show that the peak throughput of nonpersistent messages is approximately the same comparing Version 5.2 to Version 5.3, with the advantage of improved scalability when using as few as 5 driving applications (throughput up by 6% : 1,863 cf. 1,966 RT/s). Using 20 driving applications throughput is improved by 10% : 1,713 cf. 1,879 RT/s).

2.2.2 Persistent messages – client channels

Queue manager log configuration:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

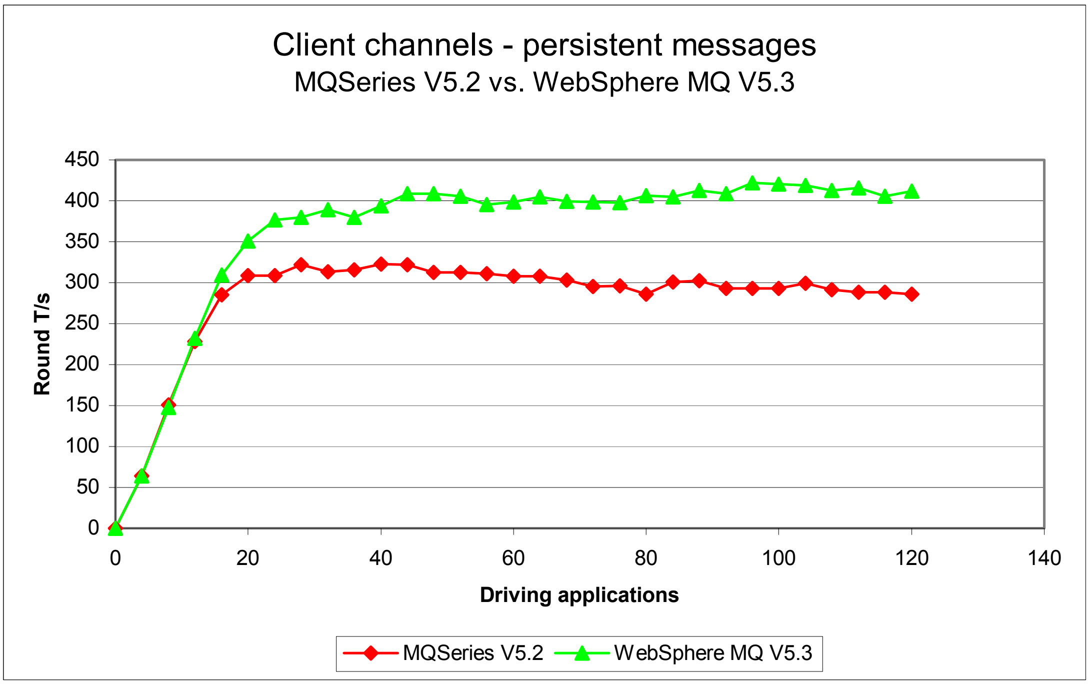

Figure 11 – Performance headline, persistent messages, client channels

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

Test name	Product version	Apps	Round T/s	%	Resp time (s)
clpm3 (inetd)	MQSeries V5.2	**40** (96) (120)	**323** (293) (286)	n/a	0.146 (0.345) (0.429)
clpm3 (inetd)	WebSphere MQ V5.3	(40) **96** (120)	(393) **422** (412)	(+22) +44 (+44)	(0.124) 0.278 (0.341)

Table 4 – Performance headline, persistent messages, client channels

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

Figure 11 and **Table 4** above show that the peak throughput of persistent messages has increased by 31% (323 cf. 422 Round T/s) comparing Version 5.2 to Version 5.3, with the advantage of improved scalability giving the most performance improvement using one hundred or so driving applications (throughput up by 44% : 293 cf. 422 RT/s).

2.2.3 'runmqlsr' vs. inetd 'amqcrsta' listener – 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. These tests also compare the difference between *nonthreaded* channels (the 'amqcrsta' process started by inetd) with *threaded* channels (the 'runmqlsr' process started by the user).

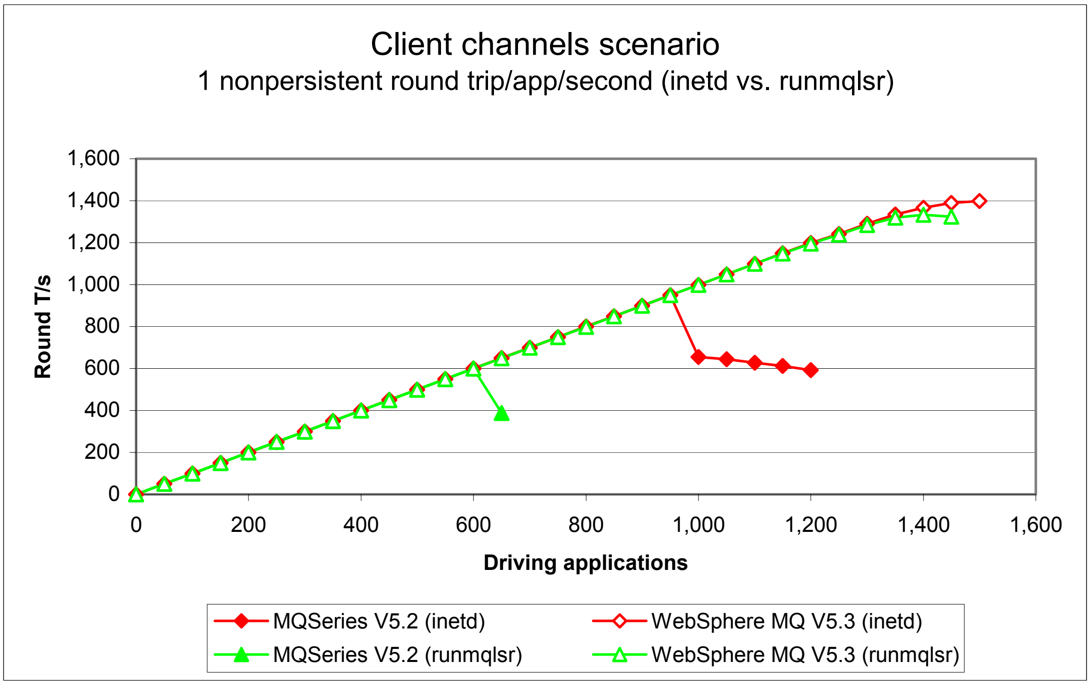

Figure 12 – 'runmqlsr' vs. inetd 'amqcrsta' listener, client channels

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

Figure 12 above and **Table 5** below show how the 'runmqlsr' and inetd 'amqcrsta' listeners in WebSphere MQ V5.3 give improved scalability by permitting a larger number of MQI-client connections into a single queue manager. In Version 5.2, either the 'runmqlsr' listener physically breaks (a well know and documented problem), or the 'amqcrsta' listener response time exceeds one second.

In Version 5.3 it is now possible to connect more than 500 or so driving application into a single queue manager (17,500 fastpath MQI-client connections: refer to '**Capacity limits – client channels**'—**Page 3**). Furthermore, the 'runmqlsr' has a reduced resource utilisation (one thread per connection compared to a process per connection for the 'amqcrsta' listener, a smaller memory footprint, less System V IPC), so is now the **preferred** method of running client channels and server channels.

Test name	Apps	Rate/App/hr	Round T/s	%	Resp time (s)
clnp1_r3600 (inetd) (MQSeries V5.2)	1,250 (950)	3,600	1,241 (949)	+31	0.864* (0.010)
clnp1_r3600_runmqlsr (MQSeries V5.2)	1,200 (600)	3,600	1,196 (600)	+99	0.968* (0.008)
clpm3_r3600 (inetd) (MQSeries V5.2)	400 (300)	3,600	375 (206)	+82	0.987* (0.386)
clpm3_r3600_runmqlsr (MQSeries V5.2)	350 (300)	3,600	349 (187)	+87	0.182* (0.700)

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

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

* Note: in **Table 5** above, the response time per round trip is given closer to one second for Version 5.3. This does not show Version 5.3 to be worse than Version 5.2 for two reasons: for Version 5.3 there are more driving applications and hence a higher message throughput at the point of constraint, and the approach of the one second response time happens in a more controlled manner. As the one second response time is exceeded, two product versions behave as illustrated below:

clnp1_r3600 - Version 5.2			clnp1_3600 - Version 5.3		
Apps	Round T/s	Resp time (s)	Apps	Round T/s	Resp time (s)
950	949	0.010	1,200	1,199	0.044
1,000	655	1.535	1,250	1,241	0.864
1,050	644	1.955	1,300	1,290	1.142

Table 5a – driving applications vs. response time

Note: directly correlating Apps to Round T/s, with Version 5.3 the message throughput and response time peak and degrade in a more controlled manner compared to Version 5.2.

Figure 13 below shows the reduced swap reservation of an MQI-client connection comparing the inetd ‘amqcrsta’ listener to the ‘runmqlsr’ listener in WebSphere MQ V5.3.

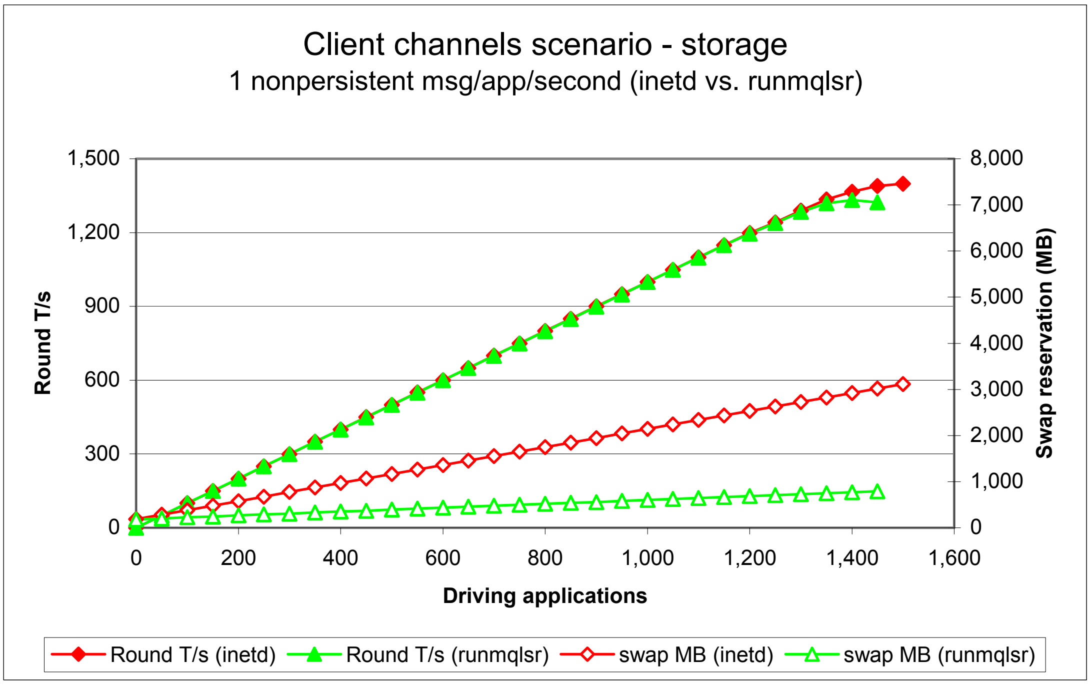

Figure 13 – ‘runmqlsr’ vs. inetd ‘amqcrsta’ listener, swap reservation, client channels

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

Test name	Apps	Swap	Free (4K pages)
clnp1_r3600 (inetd)	100 (1,000)	1.95MB/App (1.96MB/App)	258 pages/App (291 pages/App)
clnp1_r3600_runmqlsr	100 (1,000)	0.46MB/App (0.43MB/App)	44 pages/App (41 pages/App)

Table 6 – ‘runmqlsr’ vs. inetd ‘amqcrsta’ listener, swap reservation, client channels

Note: the large figures in the table above show the swap reservation and free pages measured for the given number of driving applications. The smaller figures in brackets are included to demonstrate that the increase in swap reservation is linearly proportional to the number of driving applications.

Table 6 above gives the swap reservation for the ‘runmqlsr’ and inetd ‘amqcrsta’ listeners for WebSphere MQ V5.3. For further calculations on the swap reservation and shared memory, refer to ‘**Table 17 – Client capacity, swap reservation**’—**Page 40**, and ‘**Table 19 – Distributed queuing capacity, swap reservation**’—**Page 42**.

2.2.4 Performance exercise for tuning the scenario

The reader should note that the following exercise is not restricted to the client channel test scenario, and may give more or less successful results if applied to the other two scenarios in this document. The actions taken are initially targeted at eliminating the visible activity on the queue disk.

Performance observations

The expectation of the test measurements in **Table 5** above was for the number of round trips for the ‘runmqlsr’ listener to exceed the round trips per second for the inetd ‘amqcrsta’ listener, which is anticipated because ‘runmqlsr’ has a smaller resource requirement. However, both tests constrain at approximately the same number of driving applications

(~1,200 Round T/s), when the messages arrive on the input queue more quickly than they can be processed by the available server application threads (there were spare CPU cycles because the system processors were only being utilised to 75%). This causes messages to overflow from the nonpersistent queue buffer to the queue file, which is evident in the message throughput levelling out and the response time increasing in excess of one second with each additional connection into the queue manager:

Apps	Round T/s	Resp time (s)	CURDEPTH	queue disk (kbps)
800	800	0.008	0	0
900	900	0.008	20	5
1,000	999	0.010	85	2
1,100	1,099	0.009	208	4
1,200	1,199	0.044	1,187	70
1,250	1,241	0.864	1,239	72
1,300	1,290	1.142	1,287	75
1,350	1,334	1.199	1,340	83

Table 6a – Effect of overflowing the nonpersistent queue buffer

Performance enhancement by example

A number of actions can be taken to avoid the nonpersistent queue buffer from overflowing: (with reference to ‘**Tuning the queue manager, Nonpersistent queue buffer**’—**Page 43**), and the reader will see that each action has a varying degree of success when applied in conjunction with the client channels test scenario:

1. use more server application threads to prolong the advantages of queue avoidance.
2. increase the size of the nonpersistent queue buffer to accommodate more messages.
3. use more input queues to reduce contention on a single queue resource.

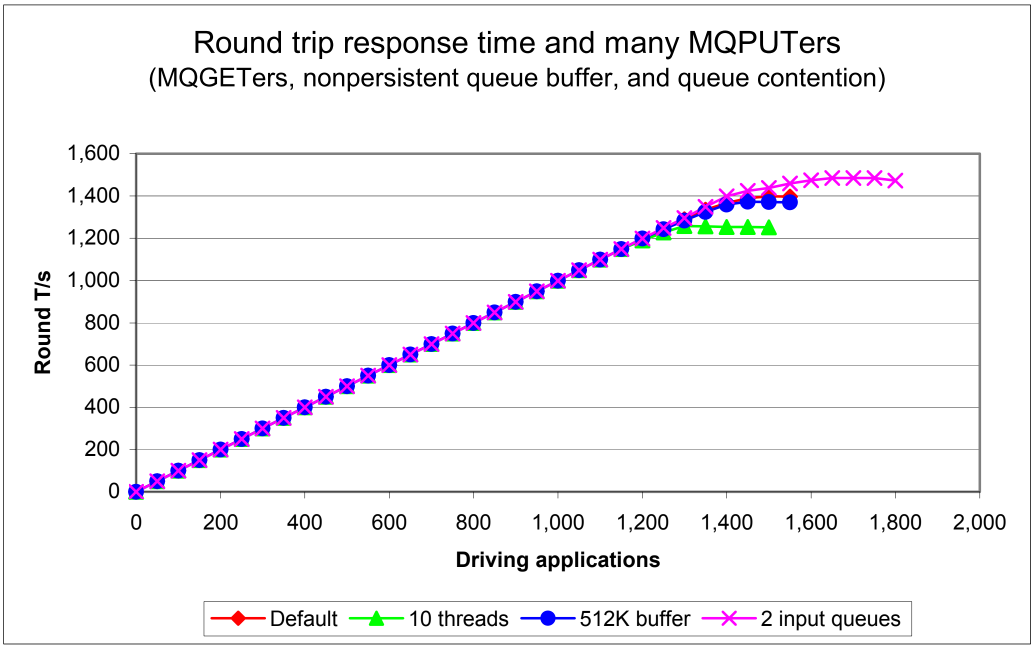

Figure 12a – Effect of number of MQGETers, size of nonpersistent queue buffer, and queues

Using more MQGETers

In line with the recommendations in '**Application design and configuration**'—**Page 45**, to process the messages on the queue quicker, additional server application threads are anticipated to assist with queue avoidance by having an MQGET call waiting on the queue when a message arrives). However, the effect of 10 server threads has resulted in excessive queue contention and throughput reaches a peak earlier than with the default number of 5 server threads (refer to '**Test scenarios workload**'—**Page 48**).

Using a bigger nonpersistent queue buffer

In line with the recommendations, a larger nonpersistent queue buffer is anticipated to assist with queue avoidance by allowing more messages to collect on the queue before being overflowed to the queue file. However, there is no significant difference in using a 512K nonpersistent queue buffer compared to the default 64K buffer. This adds weight to the '**Nonpersistent queue buffer**' discussion on **page 43**, which explains that nonpersistent queue buffer is only intended to absorb peaks in message throughput, whereas this test measurement gives a *sustained* throughput of approximately 1,400 Round T/s.

Using more input queues

In line with the recommendations, more input queues is anticipated to assist with queue contention, clearly, by spreading the incoming request messages over more than one input queue (a side effect of using more than one queue, is that each queue has its own nonpersistent queue buffer of default size of 64K). This action has the most effect on delaying the onset of the queue overflowing and associated performance degradation. The investigation of the number of input queues was extended to the tight-loop measurement for the client channel test scenario in table and figure below.

Test name	Apps	Round T/s	Resp time (s)
clnp1 Note: 1 input queue	5 (20)	1,966 (1,879)	0.002 (0.012)
clnp1_2q Note: 2 input queues	8 (20)	2,291 (2,195)	0.004 (0.011)
clnp1_4q Note: 4 input queues	12 (20)	2,513 (2,340)	0.005 (0.009)

Table 3a – Effect of number of queues on performance headline, client channels

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to show the throughput for the last measurement point.

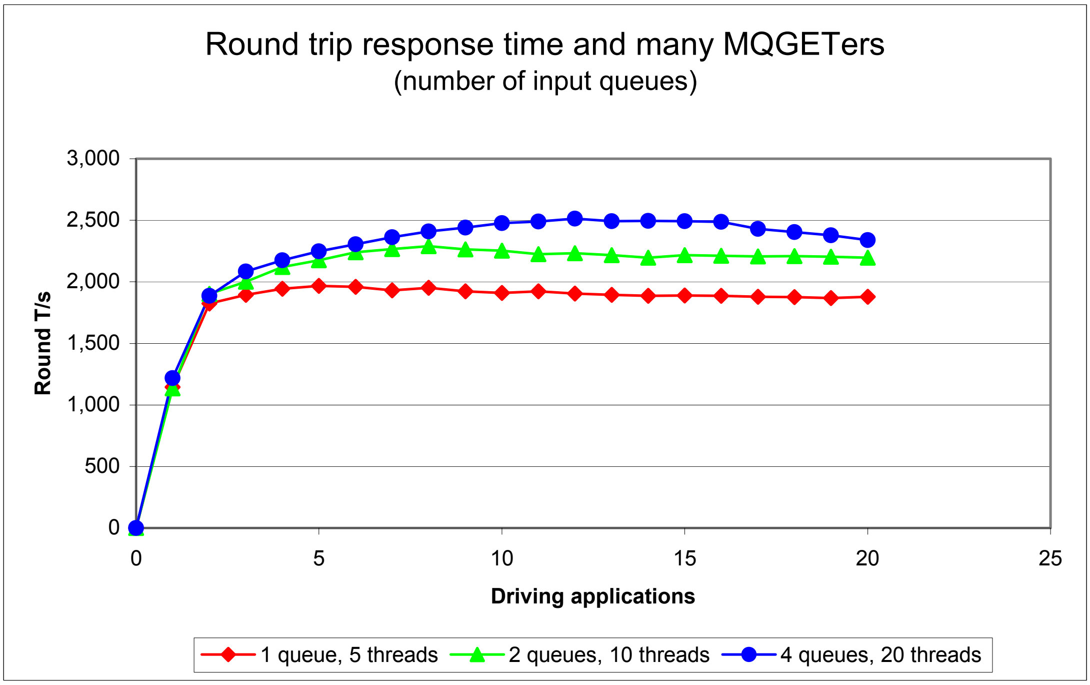

Figure 12b – Effect of number of queues on performance headline, client channels

The figure above shows how queue contention is reduced and peak throughput can be driven higher than the numbers given in '**Peak message throughput – client channels**'—**Page 2**. This is achieved using more than one input queue, with 5 threads (i.e. MQGETers) processing messages on each queue. Any processing threads over the 'optimal' number per queue will be liable to cause performance degradation through queue contention, as opposed to providing a performance enhancement—that is illustrated with the figure previous.

2.3 Distributed queuing test scenario

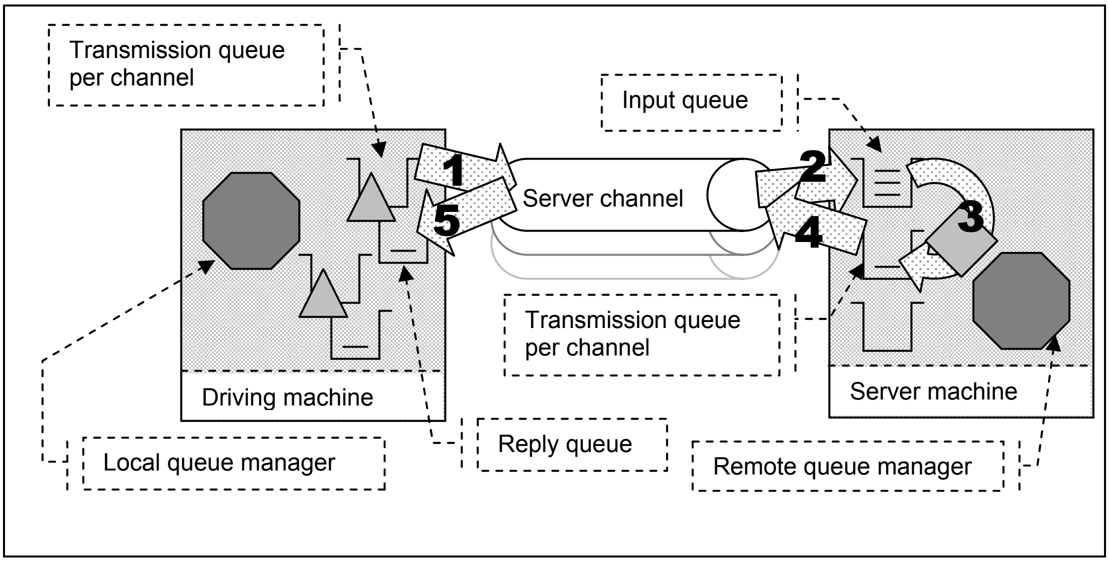

Figure 14 – Server channels between two queue managers

1) The driving 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 driving 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 server 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 driving application gets a reply from a local queue. The driving application uses the message identifier held from when the request message was put to the local definition of the remote queue, as the correlation identifier in the message descriptor.

Nonpersistent and persistent messages were used in the distributed queuing tests, with a message size of 2K. The effect of message throughput with larger messages sizes is investigated in '**Large messages: 20K and 200K – distributed queuing**'—**Page 31**.

Figure 15 and **Figure 16** below show the nonpersistent and persistent message throughput achieved using an *increasing* number of driving applications in the distributed queuing scenario (see **Figure 14** above), and WebSphere MQ V5.3 compared to Version 5.2.

2.3.1 Nonpersistent messages – server channels

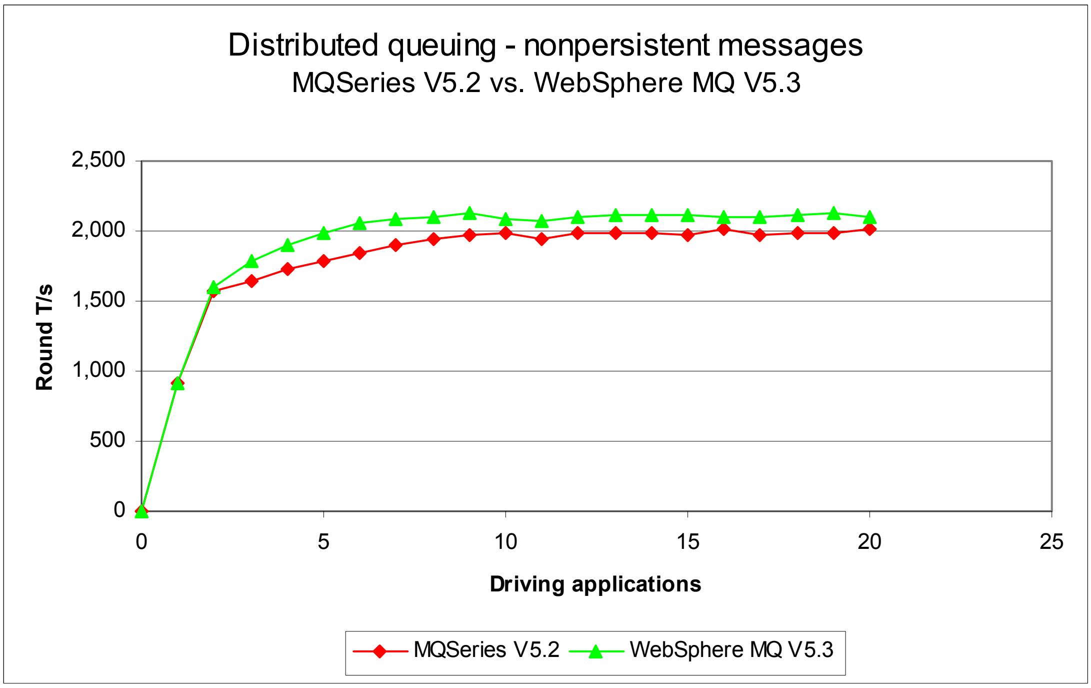

Figure 15 – Performance headline, nonpersistent messages, server channels

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

Test name	Product version	Apps	Round T/s	%	Resp time (s)
dqnp1 (inetd)	MQSeries V5.2	**16** (20)	**2,021** (2,013)	n/a	0.009 (0.011)
dqnp1 (inetd)	WebSphere MQ V5.3	**19** (20)	**2,128** (2,106)	+5 (+5)	0.010 (0.010)

Table 7 – Performance headline, nonpersistent messages, server channels

Note: the large bold figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

Figure 15 and **Table 7** above show that the peak throughput of nonpersistent messages has increased by 5% (2,021 cf. 2,128 RT/s) comparing Version 5.2 to Version 5.3, also with the advantage of improved scalability when using as few as 5 driving applications (throughput up by 11% : 1,788 cf. 1,987 RT/s). Using 20 driving applications throughput is improved by 5% : 2,013 cf. 2,106 RT/s.

2.3.2 Persistent messages – server channels

Queue manager log configuration:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

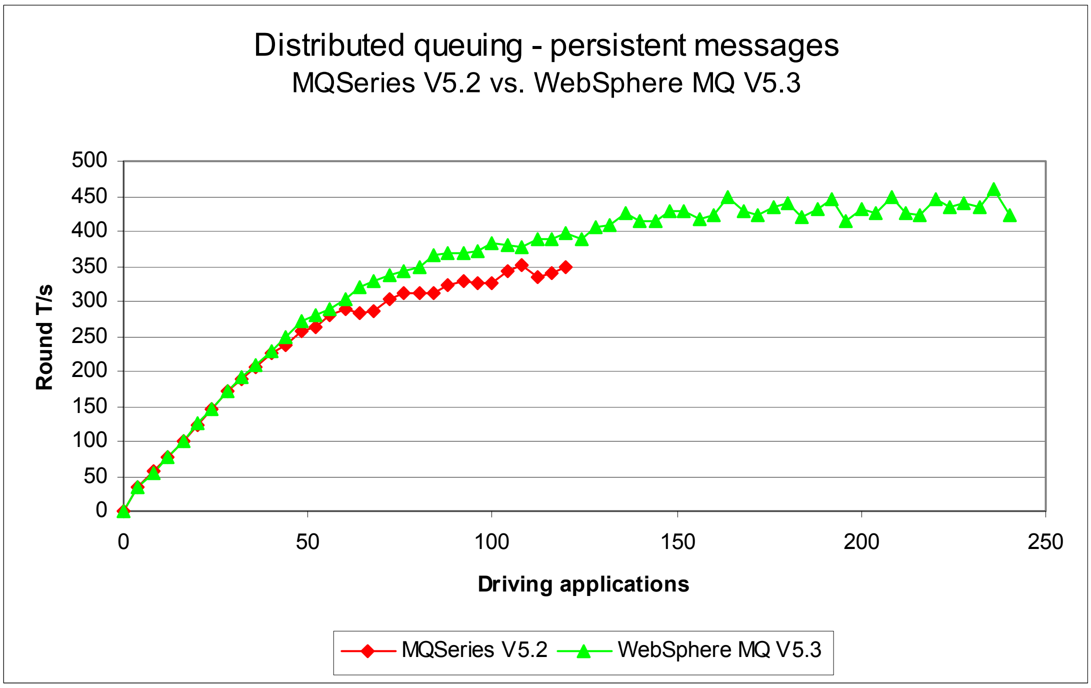

Figure 16 – Performance headline, persistent messages, server channels

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

Test name	Product version	Apps	Round T/s	%	Resp time (s)
dqpm1 (inetd)	MQSeries V5.2	108 (120)	353 (350)	n/a	0.365 (0.395)
dqpm1 (inetd)	WebSphere MQ V5.3	236 (120)	459 (408)	+30 (+17)	0.627 (0.351)

Table 8 – Performance headline, persistent messages, server channels

Note: the large figures in the table above show the peak number of round trips per second—within the range of the test, and the number of driving applications used to achieve the throughput. The smaller figures in brackets are included in the table to provide meaningful comparison between WebSphere MQ V5.3 and Version 5.2, especially since the Version 5.2 test had not constrained on response time at 120 driving applications. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

Figure 16 and **Table 8** above show that at 120 driving applications persistent message throughput has increased by 17% (350 cf. 408 RT/s) comparing Version 5.2 to Version 5.3, also with the advantage of improved scalability after 40 driving applications.

2.3.3 ‘runmqlsr’ vs. inetd ‘amqcrsta’ listener – server channels

For the following distributed queuing measurements, the messaging rate used is 1 round trip per driving application per *second*, i.e. a request message outbound over the sender channel, and a reply message inbound over the receiver channel per second. Note that there are a fixed number of 4 server channel pairs for the nonpersistent messaging tests, and 2 pairs for the persistent message tests. These tests also compare the difference between *nonthreaded* channels (the ‘amqcrsta’ process started by inetd, and the ‘runmqchl’ process started by the queue manager) with *threaded* channels (the ‘runmqlsr’ process started by the user, and the ‘runmqchi’ process started with the queue manager).

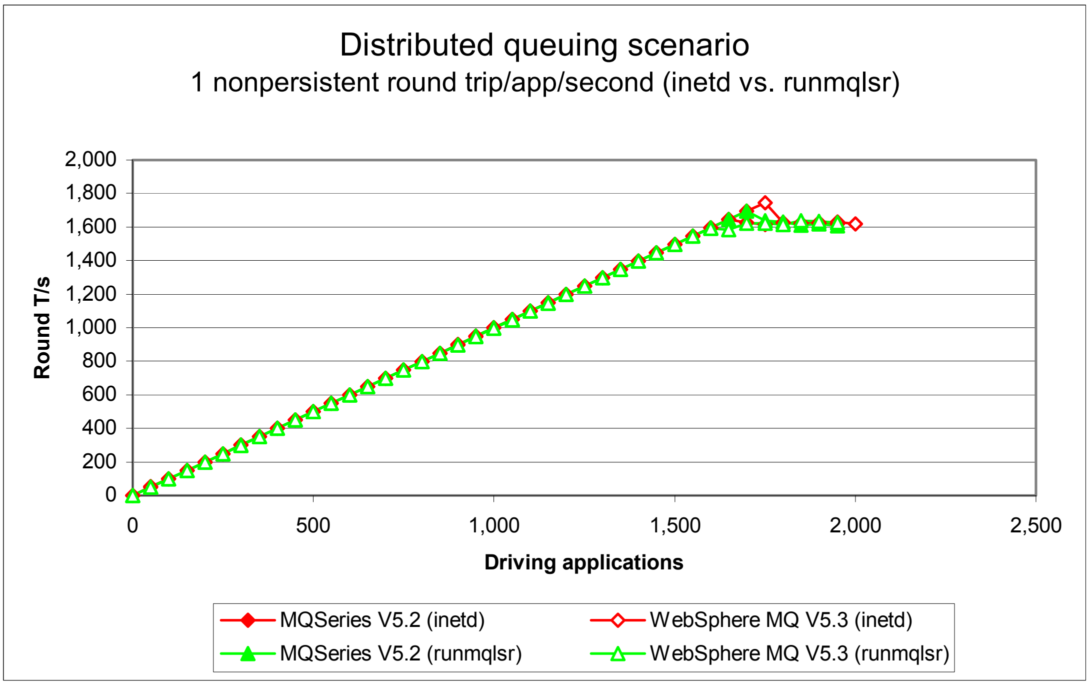

Figure 17 – ‘runmqlsr’ vs. inetd ‘amqcrsta’ listener, server channels

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

Figure 17 above shows that there is little difference between the inetd ‘amqcrsta’ and ‘runmqlsr’ listener in terms of the number of round trips that can be achieved per second before the round trip response time exceeds one second. This is likely because although the message throughput is in excess of 1,000 Round T/s, there are only 4 server channel pairs between the two queue managers, which gives little opportunity to show how ‘runmqlsr’ is more resource efficient than the ‘amqcrsta’ listener when using very few server channel pairs.

Note: it was anticipated that the WebSphere MQ V5.3 'runmqlsr' should have a reduced resource utilisation compared to the WebSphere MQ V5.3 'amqcrsta' listener and the Version 5.2 listeners; therefore, a larger number of driving applications would be able to process more messages over the server channels before the measurement constrained. However, analysis of the performance data revealed that a number of messages were building up on the input queue because the server application threads were not able to remove and process them quickly enough (there were spare CPU cycles because the system processors were being utilised to less than 60%). With reference to '**Performance exercise for tuning the scenario**'—**Page 13**, using more threads in the server program or increasing the size of the nonpersistent queue buffer is not likely to provide a performance enhancement, whereas, using more input queues is most likely to give a higher message throughput will less than a one second response time.

Test name	Apps	Rate/App/hr	Round T/s	%	Resp time (s)
dqnp1_r3600 (inetd) (MQSeries V5.2)	1,750 (1,700)	3,600	1,744 (1,644)	+6	0.017 (0.070)
dqnp1_r3600_runmqlsr (MQSeries V5.2)	1,650 (1,700)	3,600	1,585 (1,694)	-6	0.825 (0.621)
dqpm1_r3600 (inetd) (MQSeries V5.2)	500 (450)	3,600	430 (396)	+9	0.879 (0.936)
dqpm1_r3600_runmqlsr (MQSeries V5.2)	400 (400)	3,600	392 (365)	+7	0.623 (0.864)

Table 9 – 1 round trip per driving application per *second*, server channels

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

3 Large messages

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

3.1 MQI response times: 50bytes to 2MB – local queue manager

Queue manager log configuration:

```
LogPrimaryFiles=3, LogFilePages=2048
```

Figure 18, **Figure 19**, **Figure 20** and **Figure 21** below show that the response for MQPUT/GET pairs is improved for persistent message sizes from 5K to 2MB—For nonpersistent messages the response time for trusted MQPUT/GET pairs is improved for messages sizes from 50 bytes to 2MB.

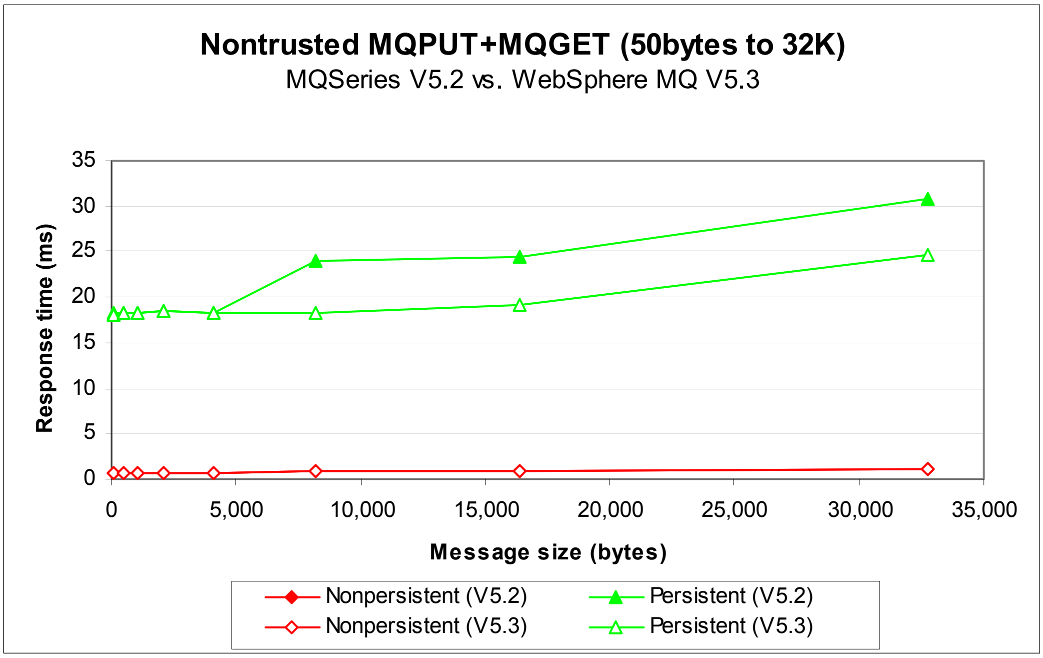

Figure 18 – Effect of message size on MQI response time (50byte - 32K)

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

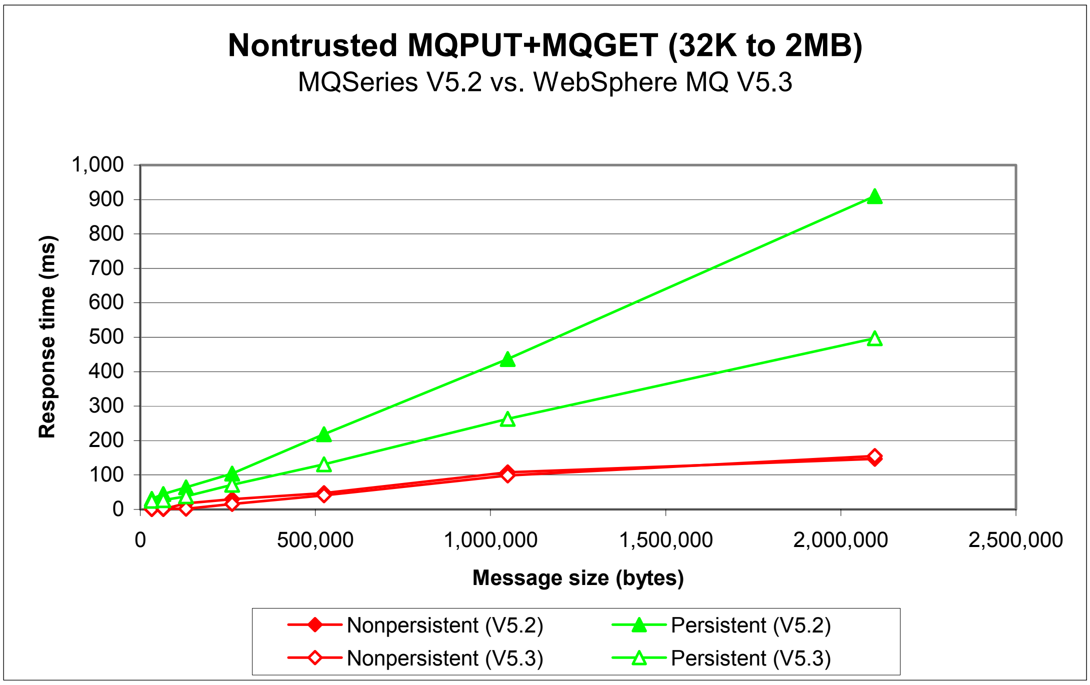

Figure 19 – Effect of message size on MQI response time (32K - 2MB)

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

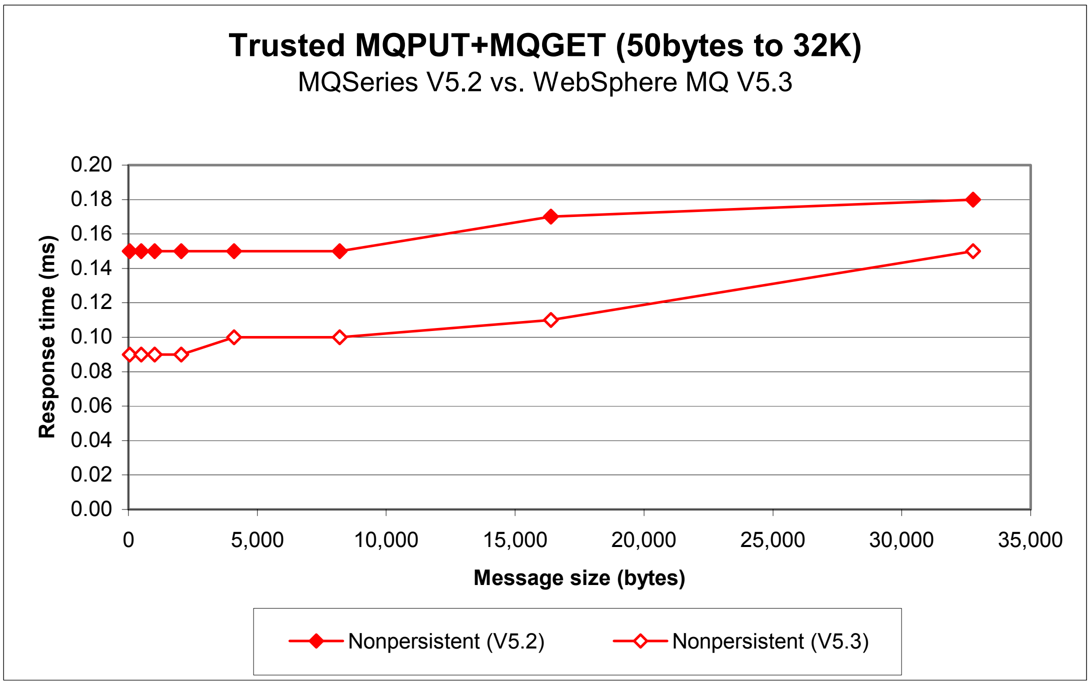

Figure 20 – Effect of message size on trusted MQI response time (50byte - 32K)

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

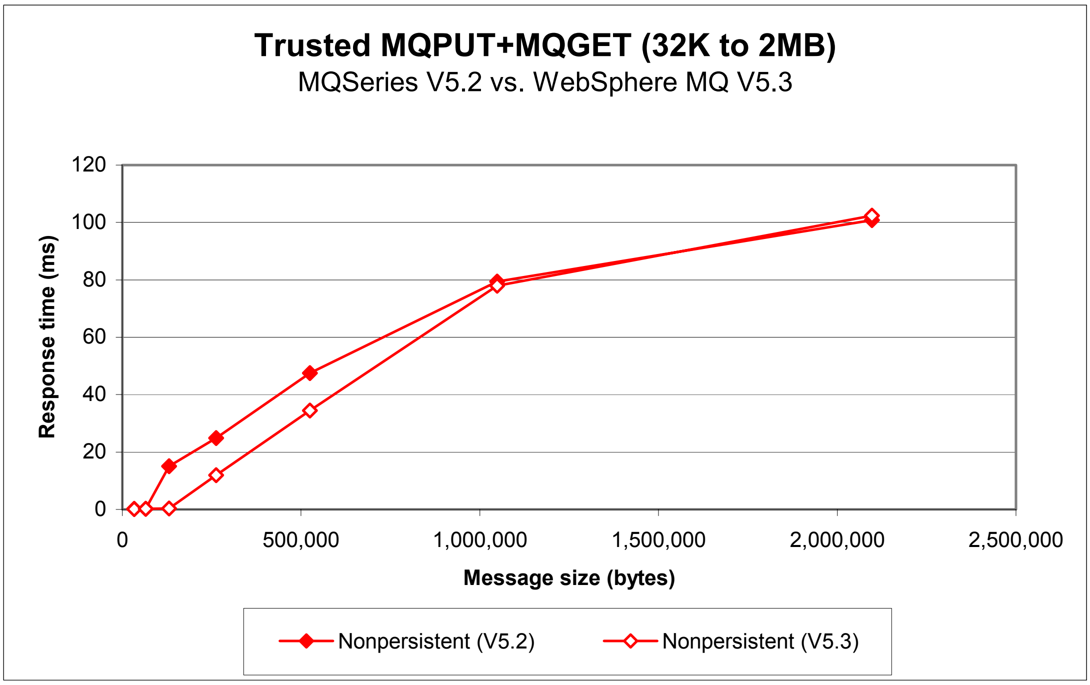

Figure 21 – Effect of message size on trusted MQI response time (32K - 2MB)

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

3.2 Large messages: 20K and 200K – local queue manager

Queue manager log configuration for persistent tests:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
local_np1_2K (MQSeries V5.2)	8 (1)	2K	2,160 (2,135)	+1	0.003 (0.001)
local_np1_20K (MQSeries V5.2)	5 (2)	20K	1,478 (1,479)	+0	0.003 (0.002)
local_np1_200K (MQSeries V5.2)	2 (3)	200K	163 (34)	+379	0.012 (0.098)
local_pm3_2K (MQSeries V5.2)	104 (28)	2K	433 (325)	+33	0.288 (0.107)
local_pm3_20K (MQSeries V5.2)	64 (44)	20K	108 (103)	+5	0.521 (0.520)
local_pm3_200K (MQSeries V5.2)	12 (8)	200K	14 (11)	+27	0.937 (0.754)

Table 10 – 2K, 20K and 200K messages, local queue manager

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

Note: the figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

The measurements for 20K and 200K *persistent* messages show that there is little difference in the performance of large messages between Version 5.2 and Version 5.3—this is because most of the time taken by the queue manager is in logging the messages to disk.

Figure 22 below shows how the throughput of small nonpersistent messages is improved significantly in WebSphere MQ V5.3, using less than 5 driving applications.

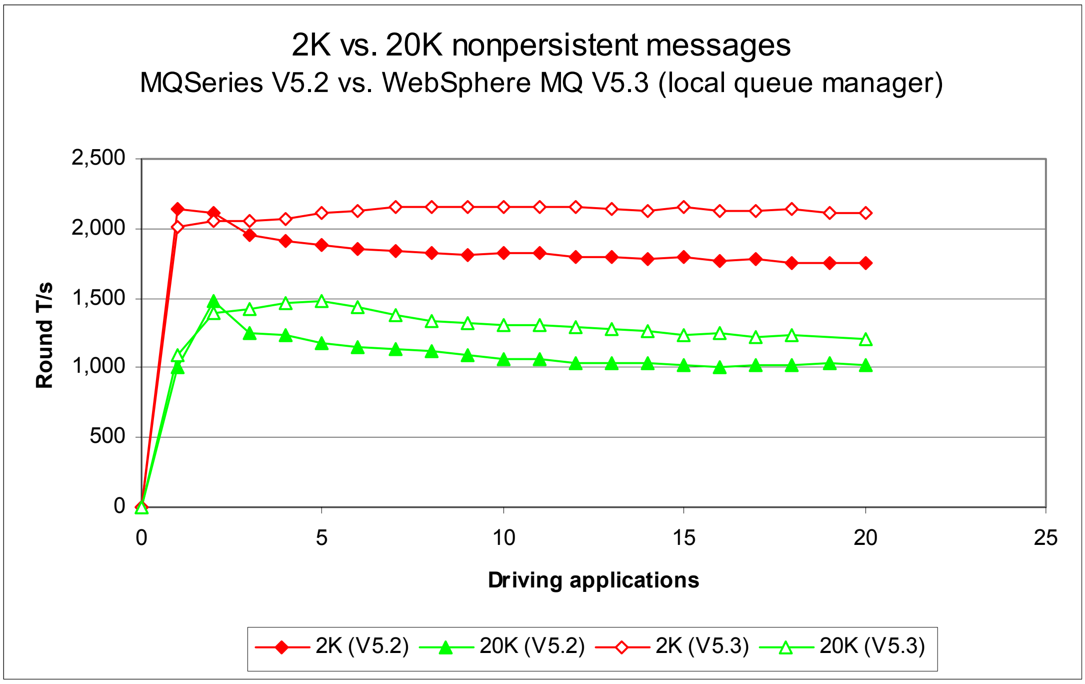

Figure 22 – 2K and 20K nonpersistent messages, local queue manager

Figure 23 below shows how the throughput of 2K persistent messages is improved significantly, and the throughput of 20K persistent messages is improved slightly, in WebSphere MQ V5.3, using more than 20 driving applications.

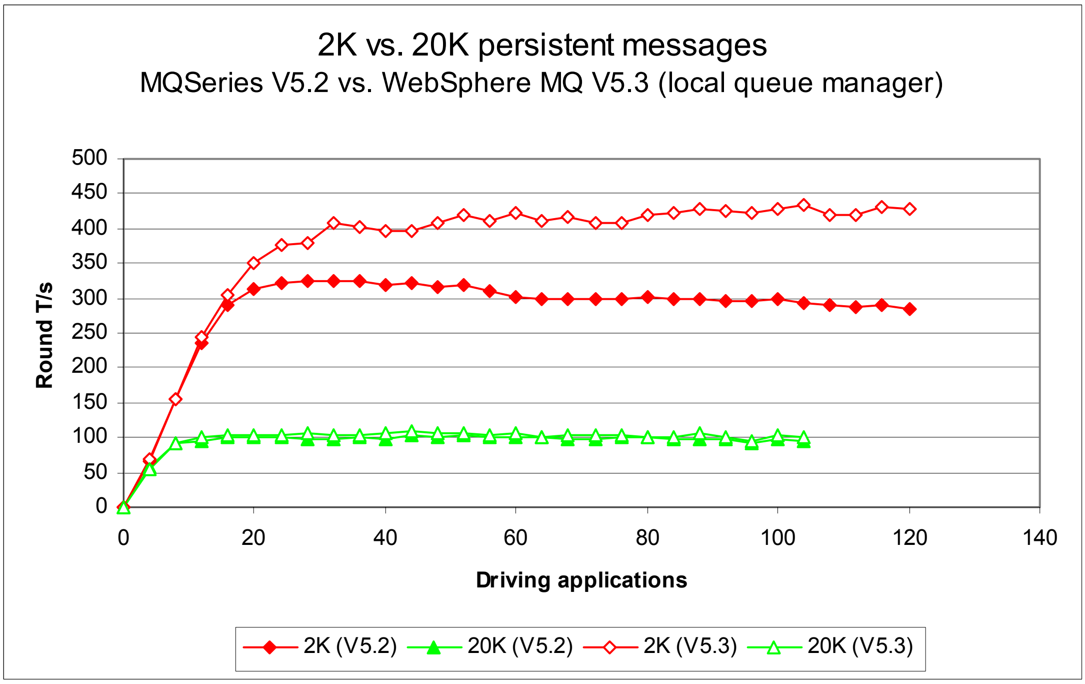

Figure 23 – 2K and 20K persistent messages, local queue manager

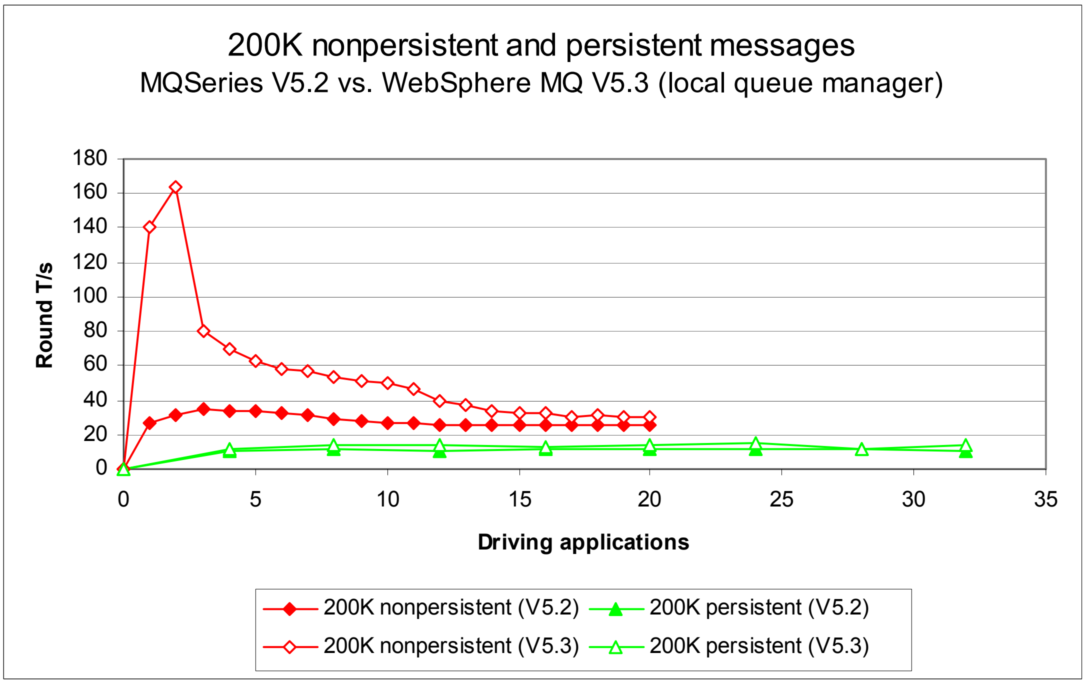

Figure 24 – 200K nonpersistent and persistent messages, local queue manager

3.3 Large messages: 20K and 200K – client channels

Queue manager log configuration for persistent tests:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
clnp1 (MQSeries V5.2)	5 (2)	2K	1,966 (1,941)	+1	0.002 (0.001)
clnp1_20K (MQSeries V5.2)	6 (6)	20K	1,105 (951)	+16	0.006 (0.007)
clnp1_200K (MQSeries V5.2)	3 (4)	200K	97 (35)	+177	0.033 (0.134)
clpm3 (MQSeries V5.2)	96 (40)	2K	421 (323)	+30	0.278 (0.146)
clpm3_20K (MQSeries V5.2)	44 (28)	20K	107 (101)	+6	0.579 (0.334)
clpm3_200K (MQSeries V5.2)	12 (8)	200K	14 (11)	+27	0.956 (0.785)

Table 11 – 2K, 20K and 200K messages, client channels

Note: messaging in these tests is with no think-time, and the inetd 'amqcrsta' listener.

Note: the figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

The measurements for 20K and 200K *persistent* messages show that there is little difference in the performance of large messages between Version 5.2 and Version 5.3—this is because most of the time taken by the queue manager is in logging the messages to disk.

Figure 25 below shows how the throughput of small nonpersistent messages is improved slightly in WebSphere MQ V5.3, using 5 or more driving applications.

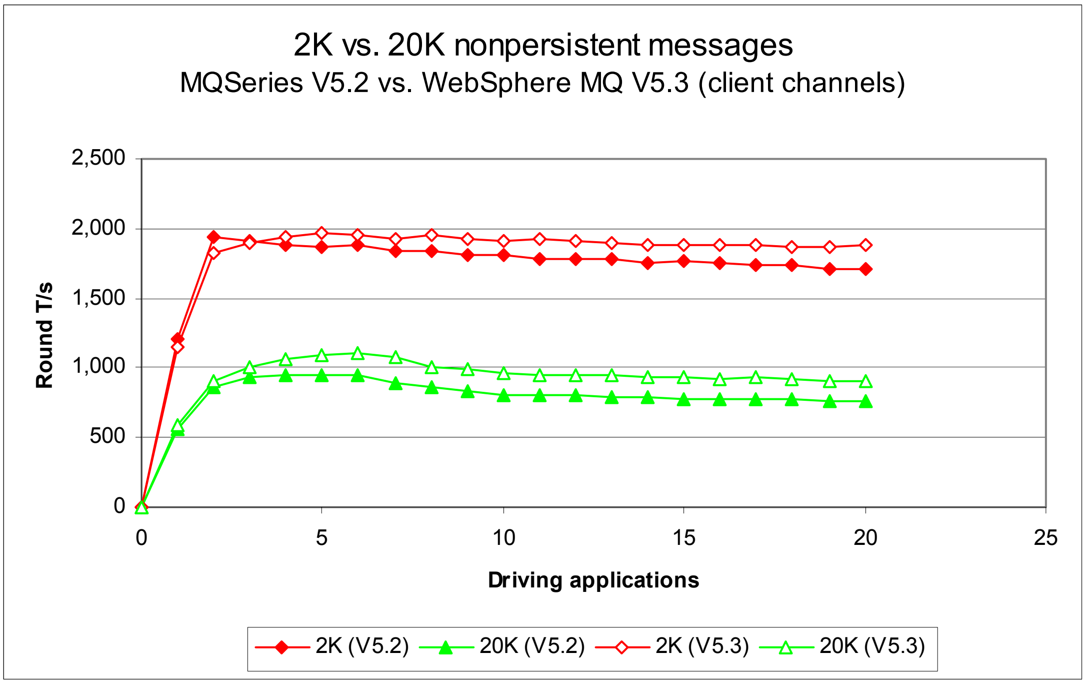

Figure 25 – 2K and 20K nonpersistent messages, client channels

Figure 26 below shows how the throughput of 2K persistent messages is improved significantly, and the throughput of 20K persistent messages is improved slightly, in WebSphere MQ V5.3, using more than 20 driving applications. Using 60 or more driving applications Version 5.3 gives a *maintained* persistent throughput of 400 round trips or more per second (33% more than Version 5.2).

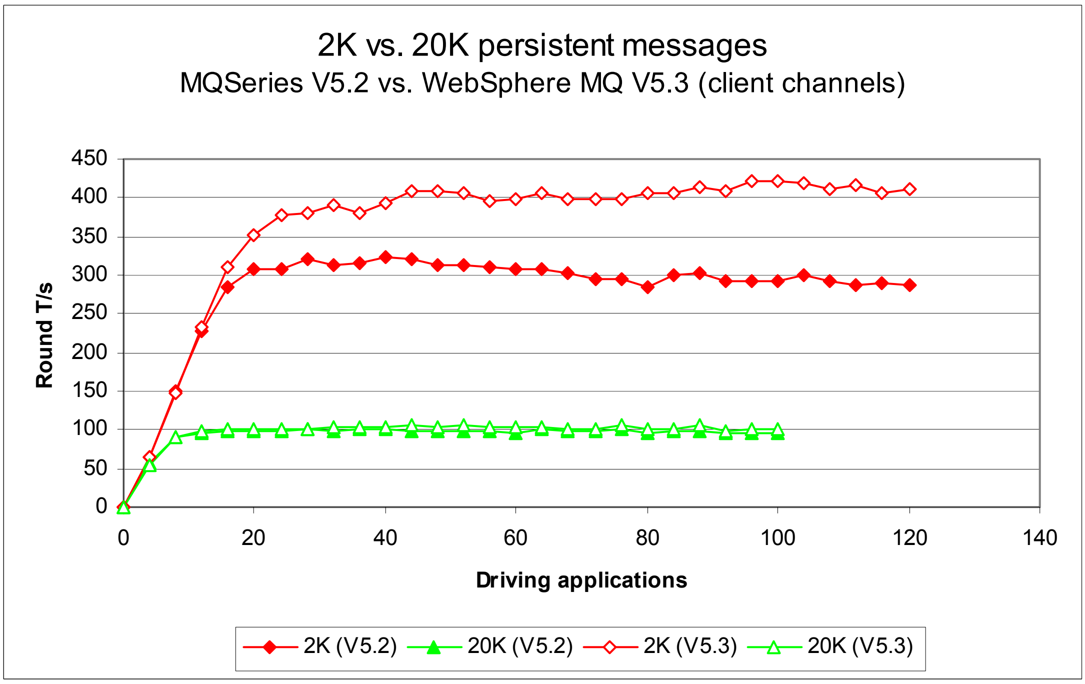

Figure 26 – 2K and 20K persistent messages, client channels

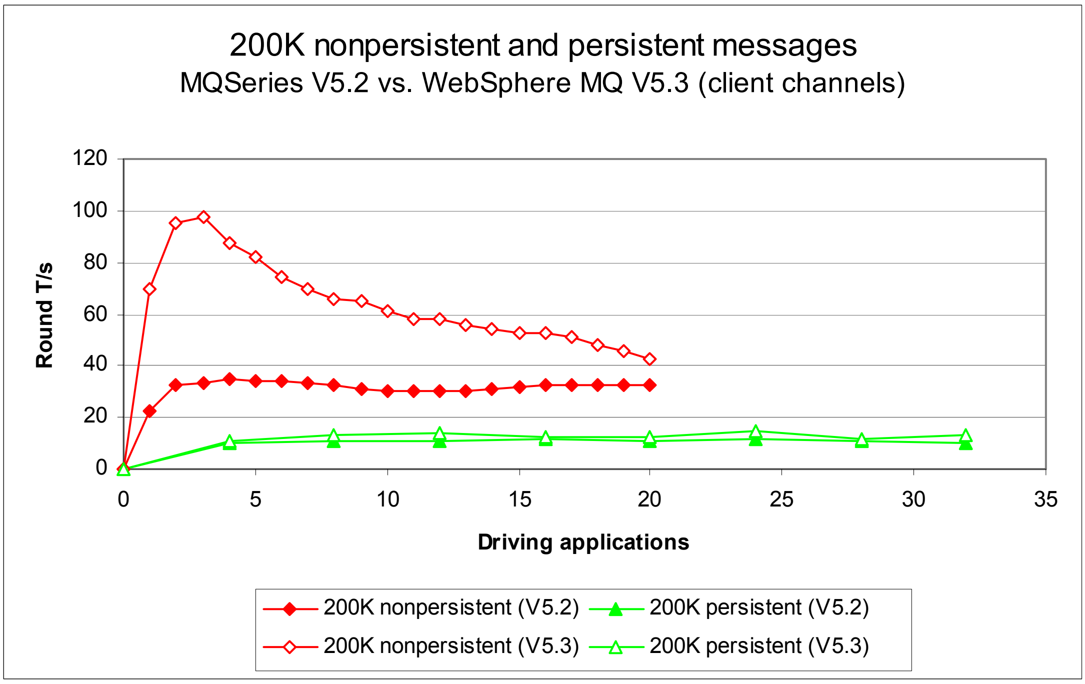

Figure 27 – 200K nonpersistent and persistent messages, client channels

3.4 Large messages: 20K and 200K – distributed queuing

Queue manager log configuration for persistent message tests:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
dqnp1 (MQSeries V5.2)	19 (16)	2K	2,128 (2,021)	+5	0.010 (0.009)
dqnp1_20K (MQSeries V5.2)	7 (10)	20K	1,076 (985)	+9	0.007 (0.012)
dqnp1_200K (MQSeries V5.2)	16 (11)	200K	83 (29)	+186	0.231 (0.440)
dqpm1 (MQSeries V5.2)	236 (108)	2K	459 (353)	+30	0.627 (0.365)
dqpm1_20K (MQSeries V5.2)	68 (56)	20K	79 (78)	+1	0.992 (0.965)
dqpm1_200K (MQSeries V5.2)	8 (4)	200K	10 (7)	+43	0.888 (0.583)

Table 12 – 2K, 20K and 200K messages, server channels

Note: messaging in these tests is with no think-time, and the inetd 'amqcrsta' listener.

Note: the figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

The measurements for 20K and 200K *persistent* messages show that there is little difference in the performance of large messages between Version 5.2 and Version 5.3—this is because most of the time taken by the queue manager is in logging the messages to disk.

Figure 28 below shows how the throughput of small nonpersistent messages is improved slightly in WebSphere MQ V5.3, using as few as 5 driving applications.

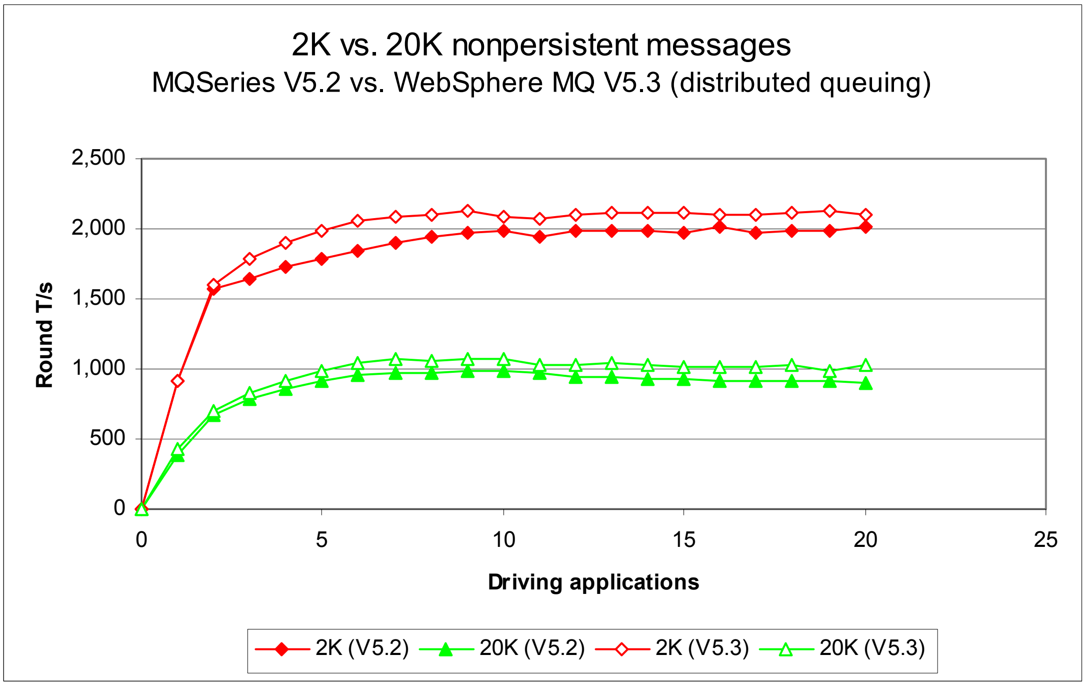

Figure 28 – 2K and 20K nonpersistent messages, server channels

Figure 29 below shows how the throughput of 2K persistent messages is improved significantly, and the throughput of 20K persistent messages is improved slightly, in Version 5.3 using more than 40 driving applications. Using more than 60 driving applications Version 5.3 gives approximately 25% more throughput (compared to Version 5.2).

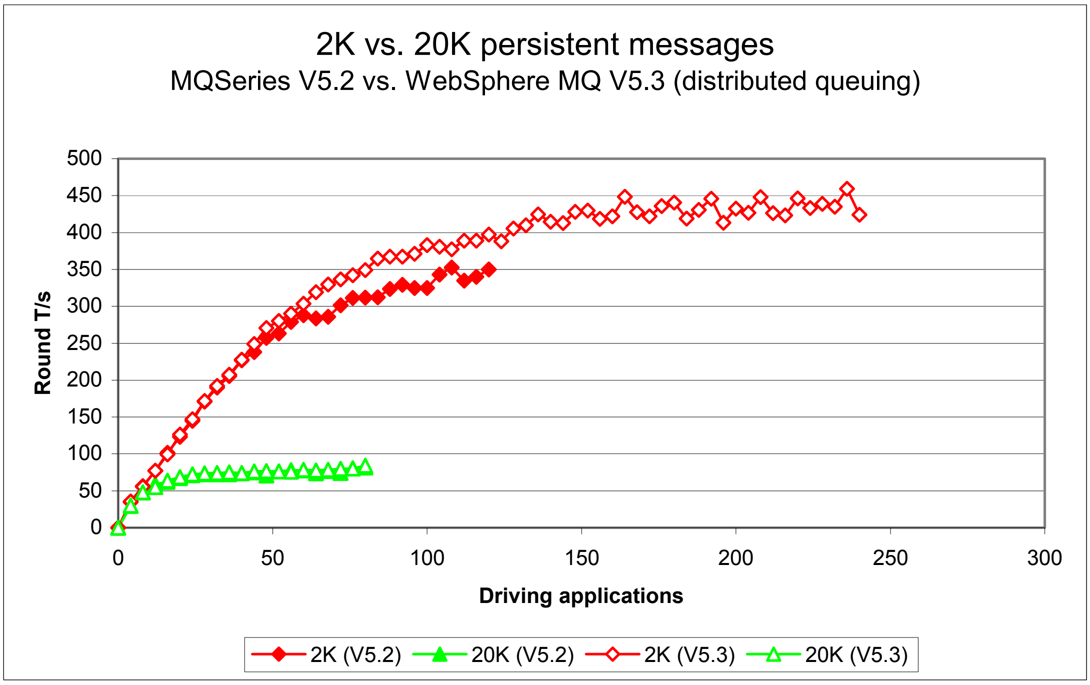

Figure 29 – 2K and 20K persistent messages, server channels

In **Figure 29** above, the 2K persistent message test for MQSeries V5.2 does not constrain on response time.

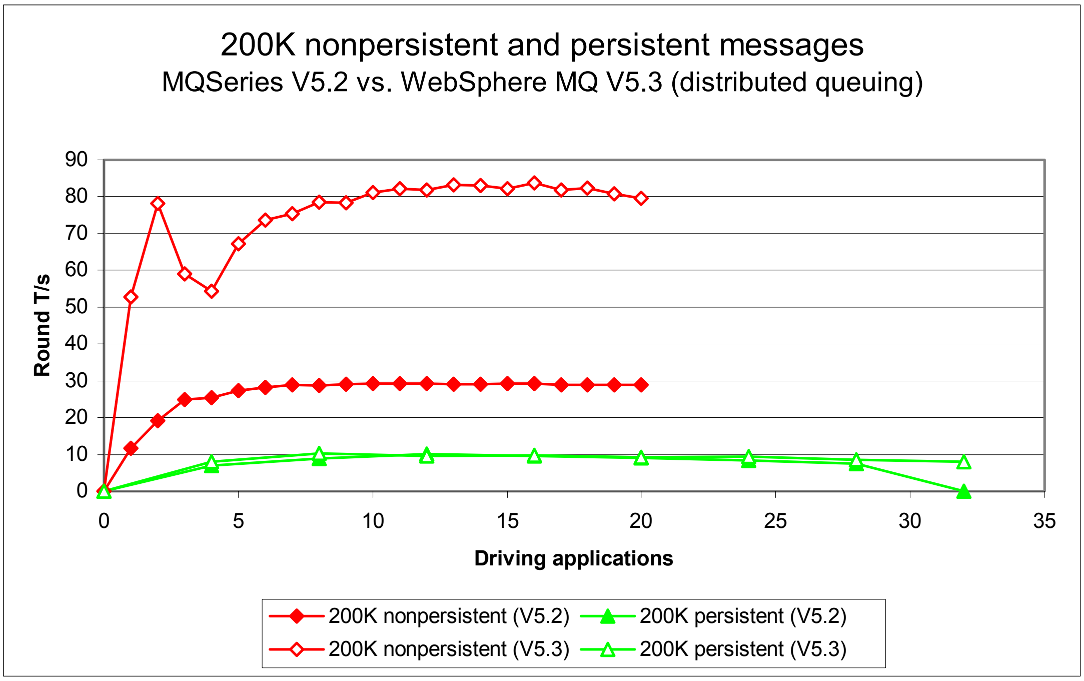

Figure 30 – 200K nonpersistent and persistent messages, server channels

With reference to **Figure 30** above, the 200K nonpersistent message tests were intentionally designed to finish at 20 driving applications. Using more than 10 driving applications, throughput neither particularly increases nor decreases. The 200K persistent message tests were designed to finish at 120 driving application, but after 4 and 8 driving application (Version 5.2 and Version 5.3 respectively) the tests approach the response time criteria.

4 Trusted server application

Queue manager log configuration for persistent tests:

```
LogPrimaryFiles=4, LogFilePages=4095, LogBufferPages=512
```

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
local_np1_trusted (MQSeries V5.2)	2 (2)	2K	4,738 (4,548)	+4	0.000 (0.001)
local_pm1_trusted (MQSeries V5.2)	120 (28)	2K	518 (380)	+36	0.297 (0.091)

Table 13 – Trusted server application, local queue manager

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

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

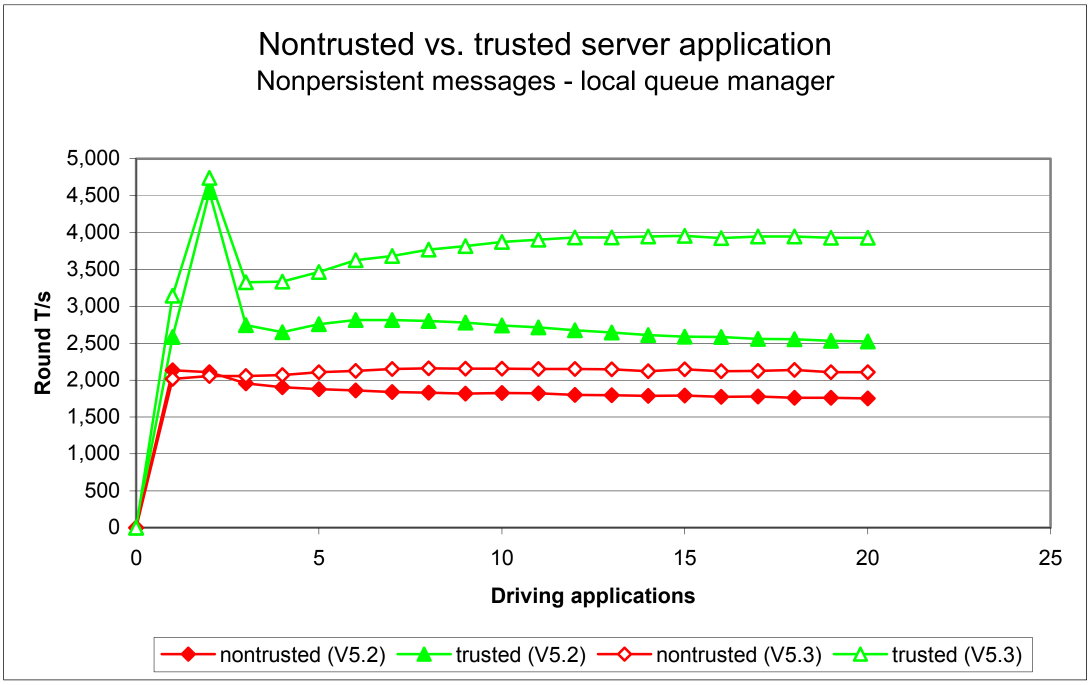

Figure 31 – Trusted server application, local queue manager

Test name	Apps	Msg size	Round T/s	%	Resp time (s)
clnp1_trusted (MQSeries V5.2)	5 (3)	2K	2,848 (2,447)	+16	0.002 (0.001)
clpm3_trusted (MQSeries V5.2)	96 (32)	2K	435 (319)	+36	0.271 (0.112)
dqnp1_trusted (MQSeries V5.2)	16 (20)	2K	3,235 (2,698)	+20	0.004 (0.007)
dqpm1_trusted (MQSeries V5.2)	188 (120)	2K	469 (353)	+33	0.547 (0.430)

Table 14 – Trusted server application, client channels and server channels

Note: messaging in these tests is with no think-time, and the inetd ‘amqcrsta’ listener.

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the percentage Round T/s improvement of Version 5.3 over Version 5.2.

5 Short sessions

A short session is a term used to describe the behavior 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,
- 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 33—Page 40** for an illustration).

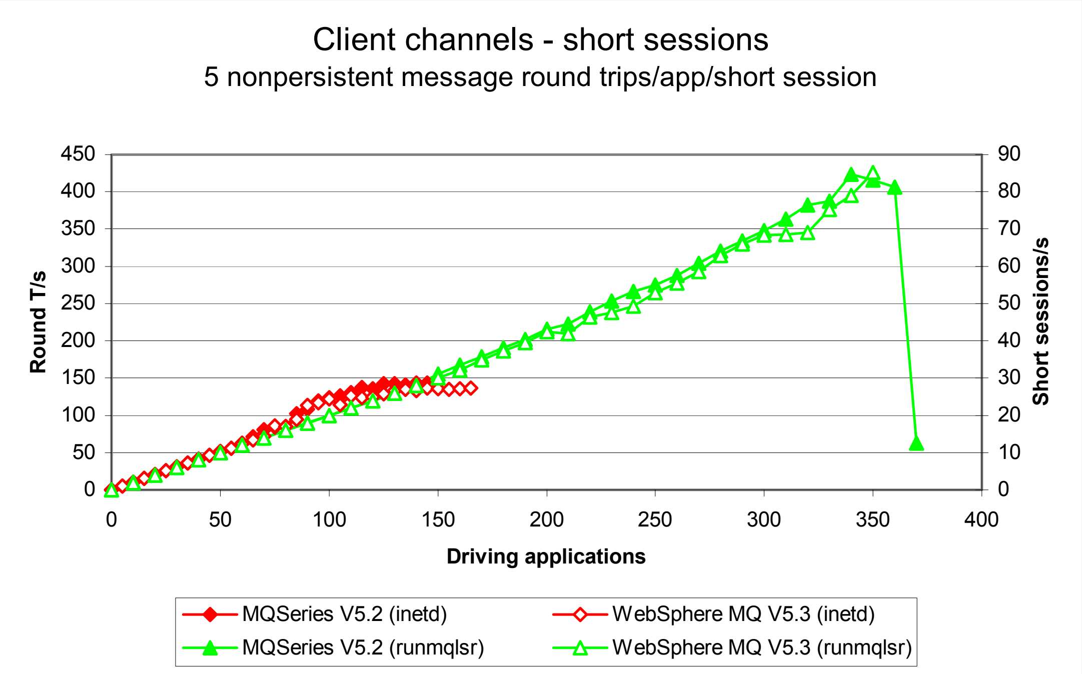

Figure 32 – Short sessions, ‘runmqlsr’ vs. inetd ‘amqcrsta’ listener, client channels

Test name	Apps	Short sessions/s	Resp time (s)	Free (4K pages)	%cpu
clnp1_ss_r3600 (inetd) (MQSeries V5.2)	145 (115)	27 (28)	0.846 (0.751)	35,324 (n/a)	93
clnp1_ss_r3600_runmqlsr (MQSeries V5.2)	350 (360)	85 (81)	0.508 (0.527)	26,461 (n/a)	53

Table 15 – Short sessions, nonpersistent messages, 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 large figures in the table above are for WebSphere MQ V5.3 with a round trip response time of less than one second. The smaller figures in brackets are for Version 5.2. Since there are 5 round trips per short session, when the round trip response time approaches a second, the short session elapsed time will be approaching 5 seconds.

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. Furthermore, in Version 5.3 the maximum number of connections into a single 'runmqlsr' listener has been significantly increased making it the **preferred** method of running short sessions over client channels.

6 Performance and capacity limits

6.1 Client channels – capacity measurements

The measurements in this section are intended to test the maximum number of MQI-clients that can be connected into a single queue manager with a message rate of 1 round trip per client channel per *minute*. In previous SupportPacs, the rate used in capacity limit tests was 1 round trip per *hour*. For the same number of client channels, a faster message rate gives a higher total message throughput over each channel. This information is intended to be more useful to the reader and assist them in projecting the results in this section to similar scenarios.

Queue manager configuration for client channel capacity tests:

```
MaxChannels=50000
```

Test name	Apps	Rate/App/hr	Round T/s	Resp time (s)	%cpu
clnp1 (inetd) (MQSeries V5.2)	5 (2)	n/a*	1,966 (1,941)	0.002 (0.001)	72 (46)
clnp1_r3600 (inetd) (MQSeries V5.2)	1,200 (950)	3,600	1,189 (949)	0.991 (0.010)	76 (85)
clnp1_c6000_t10 (inetd)	6,000	110	191	0.005	18
clnp1_c6000_runmqlsr_t10	6,000	410	688	0.182	54
clnp1_cmax_t10 (inetd) (MQSeries V5.2)	6,100 (4,800)	60 (60)	102 (80)	0.004 (0.003)	15
clnp1_cmax_runmqlsr_t10 (unique reply queue per app)	17,500 (7,500)	60 (60)	287 (171)	0.575 (0.017)	77 (12)

Table 16 – Capacity measurements, client channels

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the CPU utilisation for the measurement point.

Note: when there are a large number of MQI-clients connected to the queue manager on the server machine, the poor relative performance of the inetd ‘amqcrsta’ listener compared to the ‘runmqlsr’ listener (see **Table 16** above) is due to the excessive amount of paging the Operating System must perform to page-in the process address space for each channel as messages arrive from the driving applications.

* Since there are **1,966 Round T/s** and 3,600 seconds in one hour, the derived throughput volume per hour is calculated to be 1,966 x 3,600 = **7,077,600 round trips per hour**. The reader should note that the number of 7,100,000 round trips in one hour was not *physically* measured over the period of one hour, however, the rates of 3,600, 110, and 410 were *actual* rates set for the measurement and obtained by the queue manager system.

With reference to **Table 16** above, it is clear to see that the inetd ‘amqcrsta’ listener is the more resource intensive than the ‘runmqlsr’ listener, both in terms of the number of Round T/s that can be achieved using a single queue manager, and the CPU utilisation required to achieve the number of Round T/s. **Table 17** below supports this observation, and also indicates the additional costs associated with using more than one reply queue per driving application.

The effect of the number of client channels on maximum message throughput can be seen in **Figure 33** below.

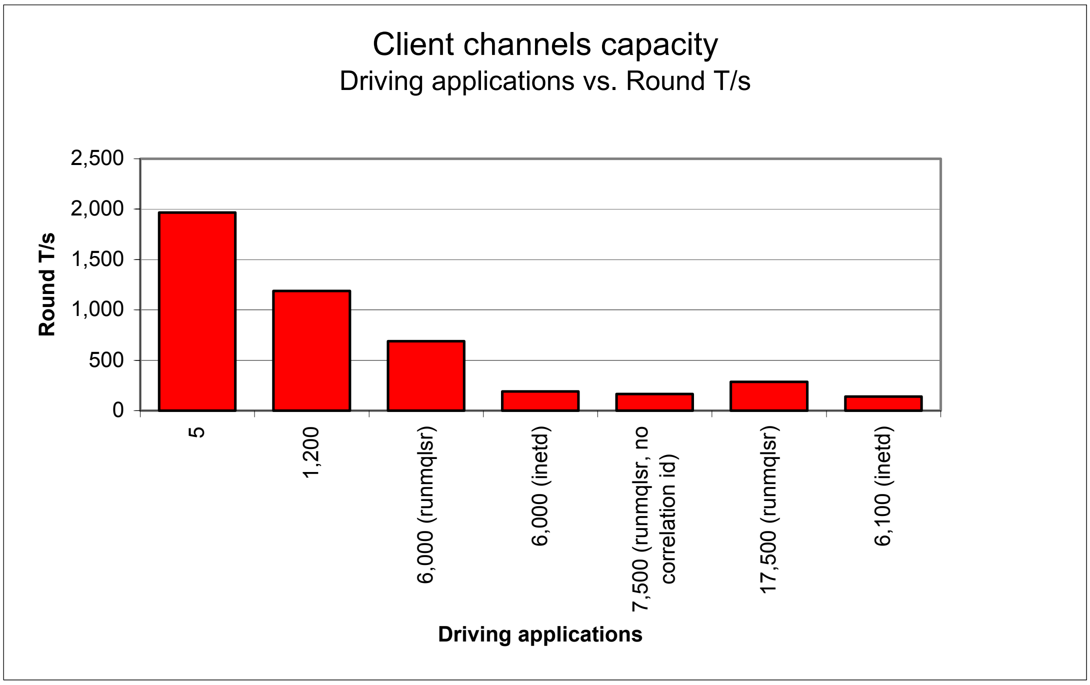

Figure 33 – Effect of number of client channels on round trips

Note: no think-time 1st column (*tight-loop*), fixed 'slow' rate 2nd column (1 nonpersistent round trip per driving application per *second*), increasing rate 3rd and 4th columns (refer to **Table 16** above for rates of *110* and *410* nonpersistent round trips per driving application per second), fixed 'medium' rate 5th, 6th and 7th columns (1 nonpersistent round trip per driving application per *minute*).

Test name	Apps	Swap	shm	Free (4K pages)
clnp1_cmax_t10 (inetd)	6,100 (1,000)	11.7GB (1.93MB/App)	44.9MB (24.0K/App) (10.2K/App)	117 pages/App (256 pages/App)
clnp1_cmax_runmqlsr_t10	17,500 (1,000)	7.1GB (0.42MB/App)	290.2MB (17.0K/App) (9.7K/App)	38 pages/App (38 pages/App)
clnp1_cmax_runmqlsr_t10 Note: unique reply queue per app	7,500 (1,000)	3.5GB (0.44MB/App)	458.9MB (62.7K/App) (32.0K/App)	48 pages/App (44 pages/App)

Table 17 – Client capacity, swap reservation

Note: the large figures in the table above show the swap reservation and shared memory measured at the given number of driving applications. The large and small figures in brackets are the proportionate swap reservation and shared memory costs per driving application (in this test scenario this relates to the cost of an MQI-client connection on the server machine).

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 pair per *minute*. In previous SupportPacs, the rate used in capacity limit tests was 1 round trip per *hour*. 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 more useful to the reader and assist them in projecting the results in this section to 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	Channel pairs	Rate/App/hr	Round T/s	Resp time (s)	%cpu
dqnp1 (inetd) (MQSeries V5.2)	19 (16)	4	n/a*	2,128 (2,021)	0.010 (0.009)	70 (74)
dqnp1_r3600 (inetd) (MQSeries V5.2)	1,750 (1,650)	4	3,600	1,744 (1,644)	0.017 (0.070)	54 (53)
dqnp1_q1000 (inetd)	1,000	1,000	2,680	712	0.895	45
dqnp1_q1000_runmqlsr	1,000	1,000	3,600	969	0.067	68
dqnp1_qmax (inetd) (MQSeries V5.2)	2,480 (2,340)	6,000	60 (60)	41 (41)	0.006 (0.007)	12 (7)
dqnp1_qmax_runmqlsr	5,000	6,000	60	83	0.146	5

Table 18 – Capacity measurements, server channels

Note: the large figures in the table above show the peak number of round trips per second, and the number of driving applications used to achieve the peak throughput. The smaller figures in brackets are included in the table to provide a comparison with MQSeries V5.2. The percentage column shows the CPU utilisation for the measurement point.

* Since there are **2,128 Round T/s**, and 3,600 seconds in one hour, the derived throughput volume per hour is calculated to be 2,128 x 3,600 = **7,660,800 round trips per hour**. The reader should note that the number of 7,700,000 round trips in one hour was not *physically* measured over the period of one hour, however, the rates of 3,600, 712, and 969 were *actual* rates set for the measurement and obtained by the queue manager system.

The fourth test in **Table 18** above does not constrain on response time. This shows that in the distributed queuing scenario used for the tests in this section (using 1,000 server channel pairs between two queue managers running on separate server machines), the two queue managers can maintain a message throughput of 1 round trip per second per server channel pair per second. This also predicates that the effective batch size over each channel is a maximum of one message.

The effect of the number of server channel pairs on maximum message throughput can be seen in **Figure 34** below.

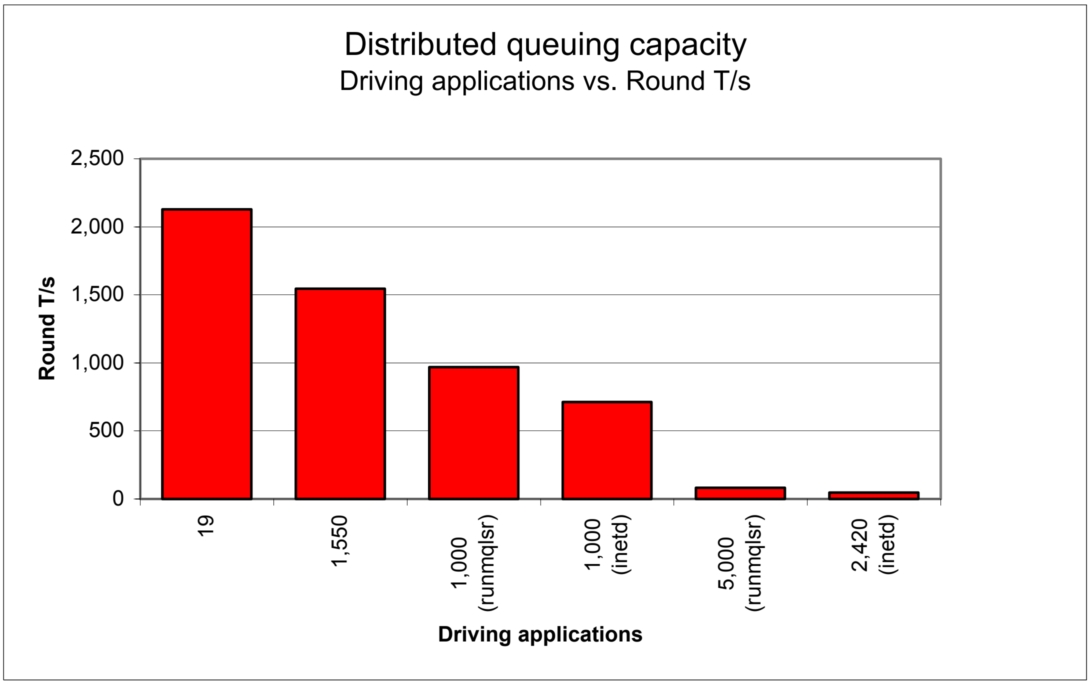

Figure 34 – Effect of number of server channels on round trips

Note: no think-time 1st column (*tight-loop*), fixed 'slow' rate 2nd column (1 nonpersistent round trip per server channel pair per *second*), increasing rate 3rd and 4th columns (refer to **Table 18** above for rates of *2,680* and *3,600* nonpersistent round trips per server channel pair per second), fixed 'medium' rate 5th and 6th columns (1 nonpersistent round trip per server channel pair per *minute*).

Test name	Apps	Swap	shm	Free (4K pages)
dqnp1_qmax (inetd)	2,420 (1,000)	10.3GB (4.20MB/App)	262.7MB (111.2K/App) (29.6K/App)	214 pages/App (514 pages/App)
dqnp1_qmax_runmqlsr	5,000 (1,000)	4.7GB (0.92MB/App)	484.1MB (99.1K/App) (28.5K/App)	92 pages/App (92 pages/App)

Table 19 – Distributed queuing capacity, swap reservation

Note: the large figures in the table above show the swap reservation and shared memory measured at the given number of driving applications. The large and small figures in brackets are the proportionate swap reservation and shared memory costs per driving application (in this test scenario this relates to the cost of a server channel pair on the server machine).

7 Tuning recommendations

7.1 Tuning the queue manager

This section highlights the tuning activities that are known to give performance benefits for WebSphere MQ V5.3; all of these can be applied equally to Version 5.2. 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 result 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 those changes that are essential. 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 may change or be removed in future releases.

7.1.1 Queue disk, Log disk, and message persistence

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

7.1.1.1 Nonpersistent queue buffer

The default nonpersistent queue buffer size is 64K per queue. This can be increased to 1MB using the TuningParameters stanza and the *DefaultQBufferSize* parameter. The nonpersistent queue buffer is computationally less expensive because the queue manager does not have to retrieve the message from the queue file. Increasing the queue buffer provides the capability to absorb peaks in message throughput at the expense of real storage, but it is **not** suitable as a long-term storage for nonpersistent messages as this buffer is **not** recovered after a queue manager restart. Defining queues using large nonpersistent queue buffers can degrade performance either if the system is short of real memory because a large number of queues have already been defined with large buffers, or for other reasons—e.g. a large numbers of channels defined.

Note: the nonpersistent queue buffer is 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 *DefaultQFileSize*. If some queues need to be defined differently to others, the values can be set in the TuningParameters stanza. When the queue manager is restarted, existing queues will keep their earlier definitions and new queues will be created with the desired parameters. 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

To improve persistent message throughput the *LogBufferPages* should be increased to its maximum configurable size of *512* x 4K pages = 2MB, the *LogFilePages* (i.e. `crtmqm -lf <LogFilePages>`) should be configured to a large size, for example: *16384* x 4K pages = 64MB, and the number of *LogPrimaryFiles* (i.e. `crtmqm -lp <LogPrimaryFiles>`) should be configured to a large number. The cumulative effect of this tuning will:

- improve the throughput of persistent messages (permitting a possible maximum 2MB of log records to be written from the log buffer to the log disk in a single write),
- reduce the frequency of log switching (permitting a greater amount of log data to be written into one extent),
- and 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 parameter takes 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 the 2MB limit of the queue manager log buffer. 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.

7.1.3 Channels: process or *thread*, standard or *fastpath*?

It is no longer necessary to consider the system design when deciding whether it is preferable to configure inetd to use process channels (‘amqcrsta’, and for server channels an MCATYPE of ‘PROCESS’), or use threaded channels (‘runmqlsr’, and for server channels an MCATYPE of ‘THREAD’) where prior to Version 5.3, it was necessary to use more than one ‘runmqlsr’ listener using more than one port. ‘runmqlsr’ **can** now **be used** in **all** scenarios with client and server channels. Additional resource savings are available using ‘runmqlsr’, including a reduced requirement on: virtual memory, number of processes (***nproc***, and ***ncallout***—refer to ‘**Tuning the operating system (HP-UX v11)**’—**Page 46**), file handles (***nfile***), and System V IPC (***shmmax***, ***semmni***, and ***shmseg***).

Fastpath channels, and/or fastpath applications—see later paragraph for further discussion, can increase throughput for both nonpersistent and persistent messaging. For persistent messages, the improvement is only for the path through the queue manager, and does not affect performance writing to the log disk. The reader should note that since the greater proportion of time for persistent messages is in the queue manager writing to the log disk, the performance improvement for fastpath channels is less apparent with persistent messages than with nonpersistent messages.

7.2 Application design and configuration

7.2.1 *Standard* or fastpath?

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

7.2.2 Parallelism, batching, and triggering

An application should be designed wherever possible to have the capability to run *multiple instances* or *multiple threads* of execution. Although the capacity of a multi-processor (SMP) system can be fully utilised with a small number of applications using nonpersistent messages, more applications are 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 batch of persistent messages to the log disk. The performance profile of a workload will also be subject to variability through cycles of low and high 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 nonpersistent 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 nonpersistent 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 nonpersistent messages are being placed on a queue quicker than they can be removed. The first effect is that messages begin to fill the nonpersistent 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 nonpersistent queue buffer, it may not be possible to avoid a performance degradation.

Processing 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—outside of syncpoint control, the queue manager must wait for each message to be written to the log before return control to the application.

A typical triggered application follows the performance profile of a short session (refer to '**Short sessions**'—**Page 37**). The 'runmqlsr' has a much smaller overhead of connecting to and disconnecting from the queue manager because it does not have to create a new process. Furthermore, in Version 5.3 the maximum number of connections into a single 'runmqlsr' listener has been significantly increased making it the preferred method of running short sessions over client channels. Nevertheless, the implementation of triggering is still worth consideration with regard to programming a disconnect interval as an input parameter to the application. 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 (HP-UX v11)

7.3.1 Number of kernel threads: 'nkthread'

The maximum number of kernel threads configurable on HP-UX v11 is 30,000. This imposes an upper-limit of 10,000 fastpath connections into a single HP-UX queue manager using 'amqcrsta' (three threads required per connection). It is recommended to reduce the overhead of each connection to 1 kernel thread and use 'runmqlsr'. The 'runmqlsr' listener in Version 5.3 permits more than 10,000 fastpath connections into a single queue manger, and does not predicate configuring more than one 'runmqlsr'.

7.3.2 Number of connections: 'nkthread', 'nproc', 'maxuprc', 'nfile'

To achieve the maximum attainable connections into a single HP-UX v11 server, the value of ***nkthread*** should be set at the maximum configurable value of ***30,000***. The value of ***nproc*** (***ncallout***), and ***maxuprc*** should also be set to: ***10,000***+ for fastpath connections into 'amqcrsta', or: 30,000 ÷ 50 = ***600+*** for fastpath connections into 'runmqlsr'—the number of '50' is dictated by the internals of the queue manager in sharing out connections between the channel pooling 'amqrmppa' processes. The ***nfile*** kernel parameter should be configured to greater than: 10,000 x (1 + 3) = ***40,000+*** for the 'amqcrsta' listener (a socket descriptor for the channel, stdin, stdout, and stderr for each 'amqcrsta' process), or 10,000 + 600 x 3 = ***28,000+*** for the 'runmqlsr' listener (a socket descriptor for the channel, stdin, stdout, and stderr for each 'amqrmppa' process).

Note: other queue manager processes require file handles such as tracing and recording FFSTs.

It is recommended that in a production environment, not as many connections are attempted into a single queue manager as documented here. The system is critically short of real memory and attempting to service a peak of message throughput could result in any queue manager process being (temporarily) deactivated by the Operating System paging daemons.

7.3.3 Swap area reservation: 'maxswapchunks'

The maximum number of fastpath client channels achieved with 4GB of real memory and 'runmqlsr' was 17,500. The total swap area reserved at 17,500 connections was approximately 7GB. The other processes running at this time were for the Operating System and queue manager. Since the greatest proportion of swap area reserved is attributable to the client channels, the swap area required per client channel is simplified by dividing the total amount of swap area reserved by the number of client connections, thus:

7GB ÷ 17,500 = 423K of swap area per fastpath client channel

The HP-UX v11 Operating System necessitates that the Virtual Set Size (VSS) of a process or thread must be reserved on swap before the process or thread can start, or before it can allocate more memory at runtime. The kernel parameters maxswapchunks should be sized to a value that, multiplied by the kernel parameter 'swapchunk' (the default is 2,048bytes) can accommodate all the required client connections. Furthermore, this space must also be available in the Operating System physical swap.

7.3.4 Swap area: performance issues

7.3.4.1 Use device swap **not** filesystem swap

Filesystem paging requires use of the buffer cache. The queue manager implicitly uses buffer cache; through the VxFS for nonpersistent messaging when the queue buffer spills messages to the queue disk, for persistent messages, and other administrative purposes. The higher the message throughput, the greater the demand can be on the buffer cache: system applications and the filesystem will have to compete for this resource. Allocating the swap on a device allows paging I/O to be made directly to the swap device bypassing the buffer cache. The system under test was configured with device swap on two physical disks allocated using the Logical Volume Manager.

7.3.4.2 "Spindles win prizes"

Using more than one physical swap device enables the Operating System to share paging requests between devices, which is quicker since there is less wait for I/O to complete. There is no performance gain in configuring more than one swap area on the same physical device.

7.3.5 Kernel tuning: System V IPC

The System V IPC kernel tuning parameters underlying the measurements in this document were the same ones as for the '**MQSeries for HP-UX V5.2 – Capacity Planning Guidance**'– **SupportPac 6E**. A summary of the kernel parameters are included in **Table 20** below for reference.

The reader **must** note that these parameters were may not be sufficient for 17,500 *standard* client connections into a single queue manager. Extra resource required by standard connections will be: CPU utilisation, shared memory, and semaphores for the additional agent processes. This is similar in ways to the additional System V IPC resources that are required for running with a unique reply queue per driving application in test name 'clnp1_cmax_runmqlsr_t10_no_correlid'.

Parameter Name	Quick Beginnings V5R2	Fastpath client channels	Fastpath server channels
`msgmap`	`258 (MQGTQL+2)`	`258`[1]	`258`[1]
`msgmax`	`4,096`	`8,192`[1]	`8,192`[1]
`msgmnb`	`4,096`	`16,384`[1]	`16,384`[1]
`msgmni`	`(not defined)`	`50`[1]	`50`[1]
`msgseg`	`1,024`	`2,048`[1]	`2,048`[1]
`msgssz`	`8`	`8`[1]	`8`[1]
`msgtql`	`256`	`256`[1]	`256`[1]
`semaem`	`16,384(<=SEMVMX)`	`32,767`[2]`(16,384`[1]`)`	`32,767`[2]`(16,384`[1]`)`
`semmap`	`1,026(=SEMMNI+2)`	`2,050(66`[1]`)`	`2,050(66`[1]`)`
`semmni`	`1,024(<=SEMMNS)`	`2,048(13`[3]`)(64`[1]`)`	`2,048(13`[3]`)(64`[1]`)`
`semmns`	`16,384(>=SEMMNI)`	`32,767`[2]`(128`[1]`)`	`32,767`[2]`(128`[1]`)`
`semmnu`	`2,048(<=nproc-4)`	`16,384`[2]`(30`[1]`)`	`16,384`[2]`(30`[1]`)`
`semume`	`256(<=SEMMNS)`	`2,048(10`[1]`)`	`2,048(10`[1]`)`
`semvmx`	`32,767(<=65,535)`	`32,767`[1]	`32,767`[1]
`shmmax`	`4,194,304`	`304,284,648`[3]	`507,565,032`[3]
`shmmni`	`1,024`	`200`[1]	`200`[1]
`shmseg`	`1,024`	`54`[3]`(120`[1]`)`	`49`[3]`(120`[1]`)`

Table 20 – Sizing the kernel tuning parameters for System V IPC

Test name	Apps	shm	segs	sems
clnp1_cmax_t10	6,100	143MB	35	11
clnp1_cmax_runmqlsr_t10	17,500	290MB	44	13
clnp1_cmax_runmqlsr_t10_no_correlid	7,500	459MB	54	13
dqnp1_qmax	2,420	263MB	43	11
dqnp1_qmax_runmqlsr	5,000	484MB	49	13

Table 20a – Sizing the System V IPC parameters for capacity limits

An example of kernel 'size' in view of the largest System V IPC kernel parameters:

```
# size /stand/vmunix
7708672 + 1164048 + 271303456 = 280176176
 (text)   (data)      (bss)   ≈ 280MB
```

[1] HP-UX v11 default kernel tuning parameter value.

[2] This value is purposely tuned to a value in the order of the maximum number of required connections.

[3] HP-UX v11 capacity limits measured value.

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) and MQDISC,
- MQOPEN and MQCLOSE,
- MQPUT and 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 scenarios workload

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 number of threads in each application was adjusted according to whether the test was measuring a local queue manager, a client channel, or distributed queuing scenario. This was done to reduce storage overheads on the driving system. Each driving application thread performed the sequence of actions as outlined in the test scenario illustrations in the '**Performance headlines**' starting on **page 5**.

Message size: For the release highlights and performance headlines (including rated messaging tests), a 2K message size was used. For the large message measurements a 20K and 200K message size was used.

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

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

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

8.1.2.2 The server application program

The server application is written as a multi-threaded program configured to use 5 threads for processing nonpersistent messages, and 20 or more threads to process persistent messages. Each server thread performed the sequence of actions as outlined in the test scenario illustrations in the '**Performance headlines**' starting on **page 5**.

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

8.2 Hardware

HP L-Class 2000:	Server system (device under test) / Driving applications machine
Model:	9000/800/L2000-44
Processor:	440Mhz PA-8500
Architecture:	4-way SMP
Memory (RAM):	4GB
Disk:	3 Internal Ultra2 SCSI (18.2GB ea. 1 O/S, 2 swap)
	4 External Ultra2 SCSI (9.1GB ea. 2 queues, 2 logs)
Network:	1GBit Ethernet

IBM S80:	Driving applications machine (for maximum server channel pairs test)
Model:	7017-S80
Processor:	375MHz PowerPC RS64-III
Architecture:	24-way SMP
	IBM SSA 160 SerialRAID Adapter
Memory (RAM):	32GB
Disk:	2 Internal 16Bit LVD SCSI (9.1GB ea. 1 O/S, 1 O/S + swap)
	3 SSA Logical disks (Note: /usr/samples/kernel/vmtune –c 0)
	(1 Physical SSA160, 9.1GB, 1 swap, 1 queue, 1 log)
Network:	1GBit Ethernet

8.3 Software

HP-UX O/S:	HP-UX v11.0 (B.11.00 C)
MQSeries:	Version 5.3 (B.11.530.00), and Version 5.2 (B.11.520.00)
Compiler:	c89 HP-UX POSIX-conforming C compiler

IBM S80 O/S:	AIX 4.3.3.0
MQSeries:	Version 5.3, and Version 5.2 (Note: queue manager CCSID 819)
Compiler:	C for AIX Compiler, Version 5 (5.0.1.0)

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** => nonpersistent messages **pm** => persistent messages **r3600** => 1 round trip per driving application per second **runmqlsr** => channels using the 'runmqlsr' listener (client channel test scenario, in addition to 'runmqchi' for distributed queuing test scenarios) **c6000** => 6,000 client driving applications (i.e. 6,000 MQI-client connections) **q1000** => 1,000 server channel pairs **max** => maximum number of channels (or channel pairs) **no_correl_id** => correlation identifier not used in the response messages (as each response is placed on a unique reply-to queue per driving application)
Apps	The number of driving applications connected to the queue manager at the point where the performance measurement is given
Rate/App/hr	The target message throughput rate of each driving application
Round T/s	The average achieved message throughput rate of all the driving applications together, measured by the server application
% (Round T/s)	The percentage increase in the total message throughput rate Note: the nature of the comparison is noted under each table where percentage improvements have been given
Resp time (s)	The average response time each round trip, as measured and averaged by all the driving applications
CURDEPTH	The number of messages on the input queue as a snapshot Note: runmqsc <qmname>, DISPLAY QLOCAL(<qname>) CURDEPTH
queue disk (kbps)	The queue disk kilobytes transferred per second
Swap	The total amount of swap area reservation for all processes in MB, unless otherwise specified as swap/app (i.e. swap area reservation per driving application) Note: swap area is reserved for ALL allocated virtual memory whether the process needs it, is physically using it, or not. This is enforced by the HP-UX kernel to ensure a process can use ALL its allocated swap should the need arise
shm	The amount of allocated System V IPC shared memory in MB
segs	The number of System V IPC shared memory segments
sems	The number of System V IPC semaphores

*** end of document ***