IBM MQ V9.4 for Windows Performance Report

Version 1.0 - December 2024

Sam Massey
IBM MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire

Notices

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 2024.

This edition applies to *IBM MQ V9.4* (and to all subsequent releases and modifications until otherwise indicated in new editions).

INTENDED AUDIENCE

This report is intended for architects, systems programmers, analysts and programmers wanting to understand the performance characteristics of IBM MQ V9.4. The information is not intended as the specification of any programming interface that is provided by IBM MQ. It is assumed that the reader is familiar with the concepts and operation of IBM MQ V9.4.

1 Preface

1.1 Target audience

The report is designed for people who:

- Will be designing and implementing solutions using IBM MQ v9.4 for Windows
- Want to understand the performance limits of IBM MQ v9.4 for Windows
- Want to understand what actions may be taken to tune IBM MQ v9.4 for Windows

The reader should have a general awareness of the Windows operating system and of IBM MQ in order to make best use of this report.

Whilst operating system, and MQ tuning details are given in this report (specific to the workloads presented), a more general consideration of tuning and best practices, with regards to application design, MQ topology etc., is no longer included in the platform performance papers. There is a stand-alone, platform neutral best practices paper available at the MQ performance GitHub page (along with performance reports on other platforms):

https://ibm-messaging.github.io/mqperf/MQ_Performance_Best_Practices_v1.0.1.pdf

1.2 Report features

This report includes:

- Release highlights with performance charts
- Performance measurements with figures and tables to present the performance capabilities of IBM MQ, across a range of message sizes, and including distributed queuing scenarios

1.3 Feedback

We welcome 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 IBM MQ system should be directed to your local IBM Representative or Support Centre.

Please direct any feedback on this report to smassey@uk.ibm.com.

2 Contents

Tables

Figures

3 Introduction

IBM MQ V9.4 is a long-term service (LTS) release of MQ, which includes features made available in the V9.3.1, V9.3.2, V9.3.3, V9.3.4 & V9.3.5 continuous delivery (CD) releases.

Performance data presented in this report does not include release to release comparisons, but all tests run showed equal or better performance than V9.3 release of IBM MQ.

As with all performance sensitive tests, you should run your own tests where possible, to simulate your production environment and circumstances you are catering for.

4 Release Highlights

Release highlights are listed in the MQ 9.4 documentation here:

https://www.ibm.com/docs/en/ibm-mq/9.4?topic=mq-whats-new-changed-in-940

4.1 Environment variables for tuning I/O operations that take too long.

From IBM MQ 9.4.0, three new environment variables are added to increase or decrease the threshold at which a warning message is written to the queue manager log if a slow read/write time is detected. Fine tuning with these environment variables can help with diagnosing operating system or storage system issues and reduce the number of errors that are written to the log. For more information, see AMQ_IODELAY, AMQ_IODELAY_INMS and AMQ_IODELAY_FFST.

For example, if the following 2 environment variables are set, then warnings will be written to the queue manager error log, when a recovery log write takes over 7000µs (7ms).

```
set AMQ_IODELAY_INMS=YES
set AMQ_IODELAY=7000
```

Using these values for a test where the recovery log is hosted on nfs; if we introduce a latency of 10ms on the link to the nfs server, MQ will report the long write time to the recovery log:

```
14/08/24 17:25:58 - Process(212992.4) User(mquser1) Program(amqzmuc0)
                    Host(mqperfx9.hursley.ibm.com) Installation(Installation1)
                    VRMF(9.4.0.0) QMgr(PERF0)
                    Time(2024-08-14T16:25:58.791Z)
                    ArithInsert1(4096)
                    CommentInsert1(W)
                    CommentInsert2(7000)
                    CommentInsert3(10126)

AMQ6729W: Log I/O operation (W) exceeded threshold (7000 microseconds).

EXPLANATION:
A Read (R) or Write (W) of 4096 bytes took longer than expected to complete.
This might indicate a problem with your O/S or storage system. If this occurs
frequently, queue manager performance is likely to be severely impacted.
ACTION:
Investigate cause of long I/O times in your storage provision.  If these delays
are expected in your environment, the warning threshold can be increased by
modifying the AMQ_IODELAY environment variable.
----- amqhose0.c : 106 --------------------------------------------------------
```

Note the 2nd and 3rd ‘CommentInsert’ fields in the message (highlighted in red), which show the current value of AMQ_IODELAY in micro-seconds and the length of time the write operation took that triggered this message.

Setting AMQ_IODELAY to a sensible value (determined by expected peak latency during a ‘normal’ day) enables you determine when the recovery log filesystem may have issues.

4.2 LZ4 Compression Options

With MQ V9.4, new LZ4 compression algorithms are available which are significantly faster than the ZLIB options. See the IBM Integration Community article here for the general details.

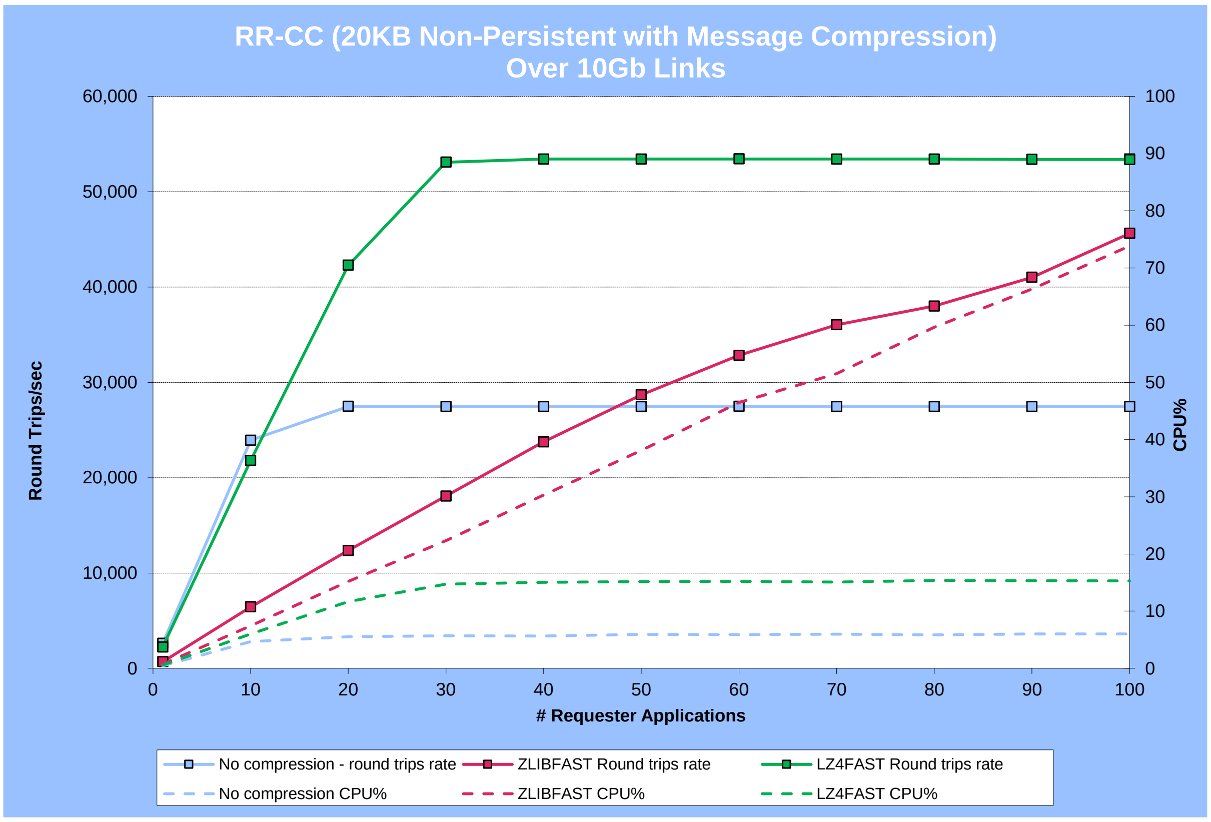

FIGURE 1- EFFECT OF MESSAGE COMPRESSION OVER NARROW BANDWIDTH NETWORK CONNECTION (10GB ETHERNET)

Results presented throughout this report were run against hosts in the same datacentre and connected via 100Gb network links. In such an environment, setting message compression is unlikely to increase the throughput, but for smaller bandwidth links, the effects can be significant.

Figure 1 above show the effects of using the ZLIBFAST or LZ4FAST compression algorithms on the SVRCONN channels used by the requester and responder applications for 20K messages over a 10Gb network. The FAST options were found to be more beneficial in this environment for both algorithms (rather than using ZLIBHIGH or LZ4HIGH). Note that the new LZ4 algorithm available in MQ V9.4 was a lot faster, consuming much less CPU and enabling a higher message rate to be achieved through the bottleneck of the 10Gb network link.

The benefit of using compression will depend on several factors including:

- Size of message
- Quality of network link (bandwidth and latency)
- Compressibility of the message data
- Available CPU resource (for both the queue manager host and the hosts where the applications are running)

Using the 10Gb linked environment used for the 20K results above, varying degrees of benefit were recorded when using compression for different message sizes.

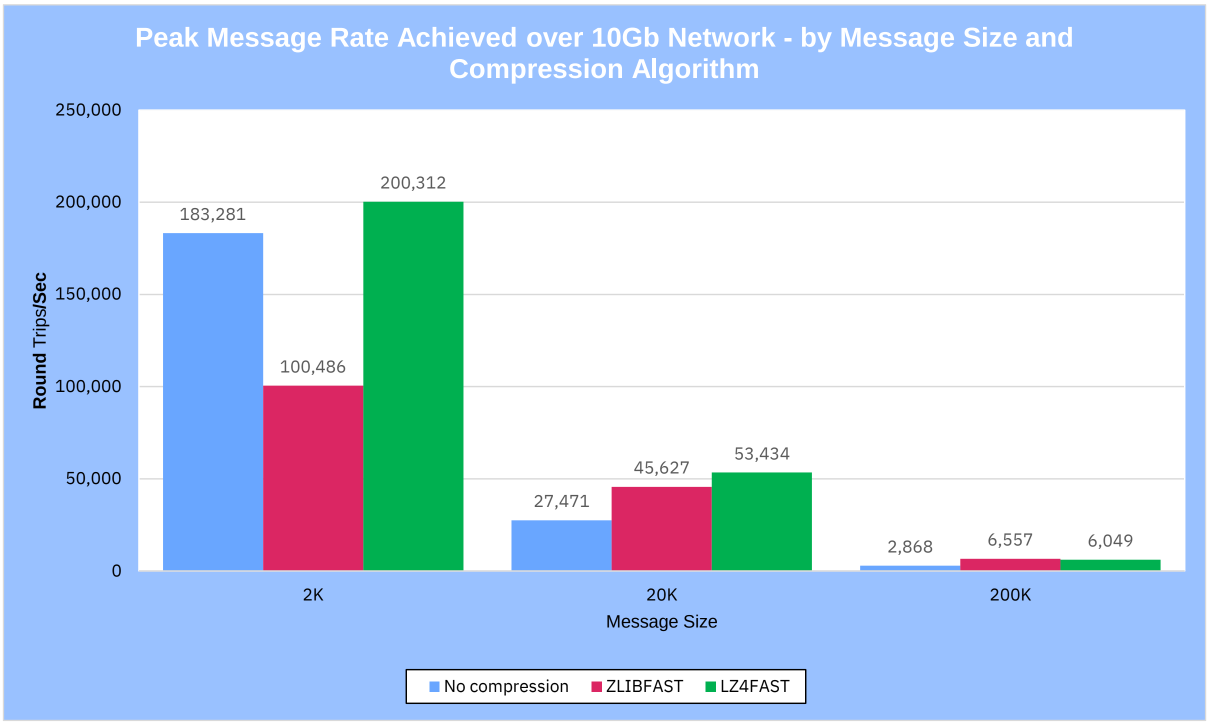

FIGURE 2- PEAK MESSAGE RATES BY MESSAGE SIZE AND COMPRESSION ALGORITHM.

Figure 2 shows the peak rate achieved for 2K, 20K and 200K messages, without compression vs compression (ZLIBFAST or LZ4FAST). Whilst the LZ4FAST always outperformed no compression, ZLIBFAST was faster for 200K messages for peak throughput.

As stated above, compressibility of the message will be a factor. For these tests, the message data was comprised of JSON text. Binary data will not benefit as much (or at all) due its uncompressible nature. Other messages may be more compressible and will benefit more.

The table below shows the compression rates achieved (as reported by the COMPRATE value in the channel status). Although the HIGH variants of the algorithms compressed these JSON messages a little more, they did not give as much benefit as the FAST variants, due to the additional time taken for the more aggressive compression.

Note that for a compression rate of 17%, the payload was compressed to 83% of its original size.

Msg Size	Compression Rate			
	ZLIBFAST	LZ4FAST	ZLIBHIGH	LZ4HIGH
2KB	34%	17%	34%	18%
20KB	56%	43%	58%	47%
200KB	59%	45%	60%	51%

TABLE 1 - MESSAGE COMPRESSION RATES

The new LZ4 compression algorithm performs significantly better than the existing ZLIB algorithm for these tests. If your environment is constrained by the network between the clients and the queue manager, it is worth setting these to test the potential benefit. ZLIB algorithms can result in higher compression rates though, so in some circumstances they may still be the best option.

Note that compression algorithms can also be set on channels between queue managers too.

4.2.1 Test setup.

Workload type: RR-CC (see section 5.2).

Hardware: Server 1, Client 1 (see section A.1).

5 Workloads

Table 2 (below) lists the workloads used in the generation of performance data for this report. All workloads are requester/responder (RR) scenarios which are synchronous in style because the application putting a message on a queue will wait for a response on the reply queue before putting the next message. They typically run ‘unrated’ (no think time between getting a reply and putting the next message on the request queue).

Workload	Description
RR-CC	Client mode requesters, and responders on hosts separate from the QM
RR-DQ-BB	Distributed queueing between two queue managers on separate hosts, with binding mode requesters and responders.

TABLE 2 - WORKLOAD TYPES

Binding mode connections use standard MQ bindings, client mode connections use fastpath channels and listeners (trusted).

RR-CC & RR-DQ-BB are described in the following sections.

5.1 Applications, Threads and Processes

From a queue manager’s perspective in the workloads described below, each connection represents a unique application. The workloads are driven by the MQ-CPH or Perfharness client emulator tools. Both these tools are multi-threaded so 10 applications may be represented by 10 threads within a single MQ-CPH process, for instance. If 200 responder applications are started, this will always be represented by 200 threads, but they could be spread across 10 processes (each with 20 threads). The main point is that each application below is a single thread of execution within MQ-CPH or JMSPerfHarness, spread across as many processes as makes sense.

5.2 RR-CC Workload (Client mode requesters, client mode responders)

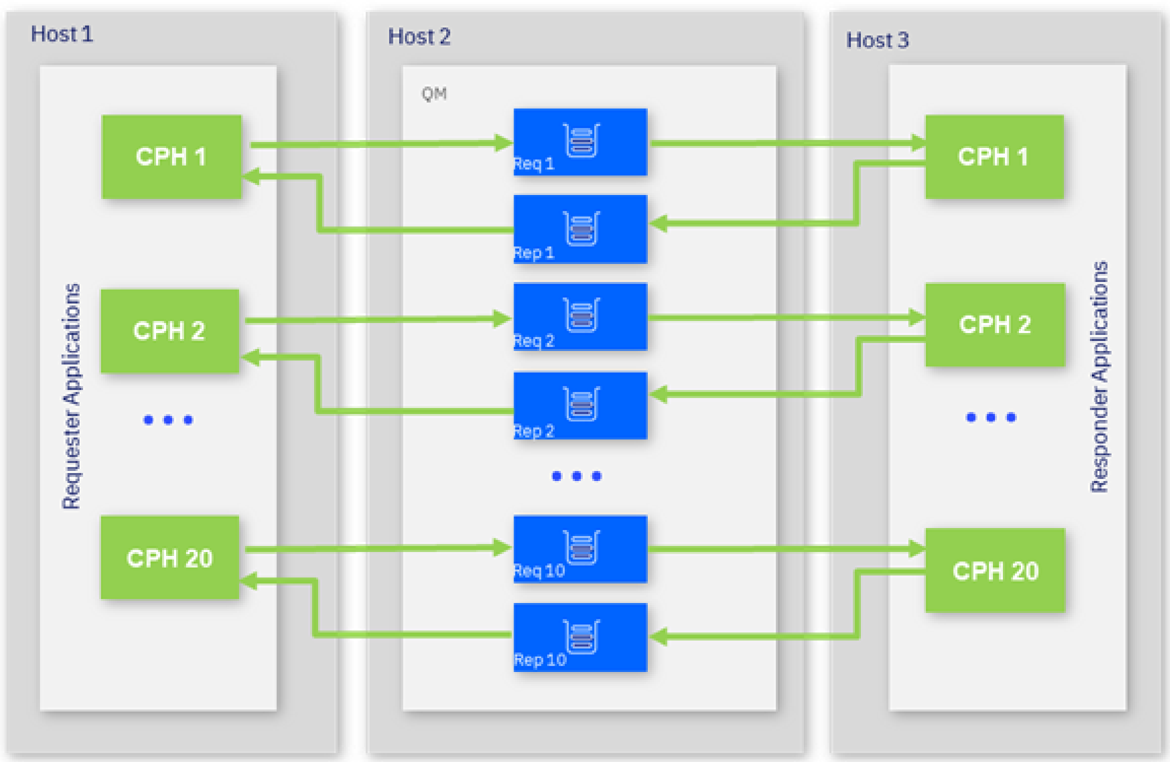

FIGURE 3 – RR-CC TOPOLOGY

Figure 3 shows the topology of the RR-CC test. The test simulates multiple 'requester' applications which all put messages onto a set of ten request queues. Each requester is a thread running in an MQI (CPH) or JMS (JMSPerfHarness) application. The threads utilise the requester queues in a round robin fashion, ensuring even distribution of traffic.

Another set of 'responder' applications retrieve the message from the request queue and put a reply of the same length onto a set of ten reply queues. Each responder is a thread of CPH or JMSPerfHarness and there may be multiple instances of these MQI or JMS applications. The number of responders is set such that there is always a waiting 'getter' for the request queue.

The applications utilise the requester and responder queues in a round robin fashion, ensuring even distribution of traffic, so that in the diagram above CPH11 will wrap round to use the Rep1/Req1 queues, and CPH 20 will use the Req10/Rep10 queues.

The flow of the test is as follows:

1. The requester application puts a message to a request queue on the remote 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 appropriate reply queue.

2. The responder application gets messages from the request queue and places a reply to the appropriate 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 reply queue using the message identifier held when the request message was put to the request queue, as the correlation identifier in the message descriptor.

This test is executed using client channels as trusted applications programs by specifying "*MQIBindType=FASTPATH*" in the qm.ini file. This is recommended generally, but not advised if you run channel exit programs and do not have a high degree of confidence in their robustness.

As the topology utilises remote clients for the requesters and responders, each round trip will comprise of 2 inbound messages to the server and 2 outbound messages from the server, all being transmitted across the network. So, if the message size is 2048 bytes there will be 2 x (2048 + metadata) inbound to the MQ server and 2 x (2048 + metadata) outbound from the server, where metadata is the non-message payload data, comprising of the MQ and transport headers.

Note for that the RR-CC scenario used in this report that the requesters and responders share a single host, remote from the Queue Manager.

5.3 RR-DQ-DB Workload (Distributed queueing between two queue managers on separate hosts, with binding mode requesters and responders).

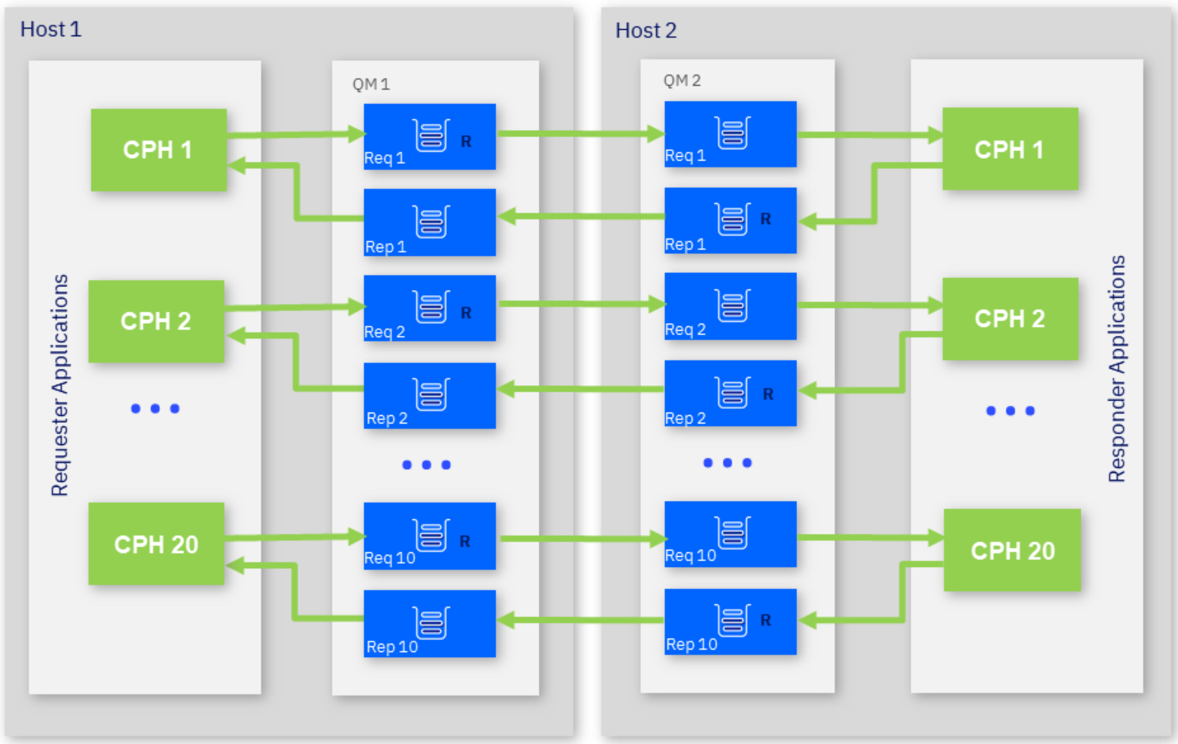

FIGURE 4 – RR-DQ-BB TOPOLOGY

This is a distributed queuing version of the requester-responder topology detailed in section 5.2. All *MQPUT*s are to remote queues (marked with 'R' in Figure 4 above) so that messages are now transported across server channels to the queue manager where the queue is hosted.

Note that remote queues are distributed across multiple pairs of sender/receiver channels in the tests below, but a single pair or channels may be adequate in your installation.

As for the RR-CC topology, each round trip will comprise of 2 inbound messages to the server and 2 outbound messages from the server, but as the clients are local to the queue manager these do not utilise network bandwidth. For each round trip there will be a single outbound and inbound message between the queue managers across the network. So, if the message size is 2048 bytes there will be 1 x (2048 + metadata) inbound to the MQ server and 1 x (2048 + metadata) outbound from the server, where metadata is the non-message payload data, comprising of the MQ and transport headers. This scenario therefore uses half the network bandwidth of RR-CC for a given message rate.

6 Non-Persistent performance test results

Full performance test results are detailed below. The test results are presented with an illustrative plot in each section followed by the peak throughput achieved for the remaining tests in that category (the remaining tests are typically for different message sizes).

6.1 RR-CC Workload

The following chart illustrates the performance of 2KB Non-persistent messaging with various numbers of requester clients.

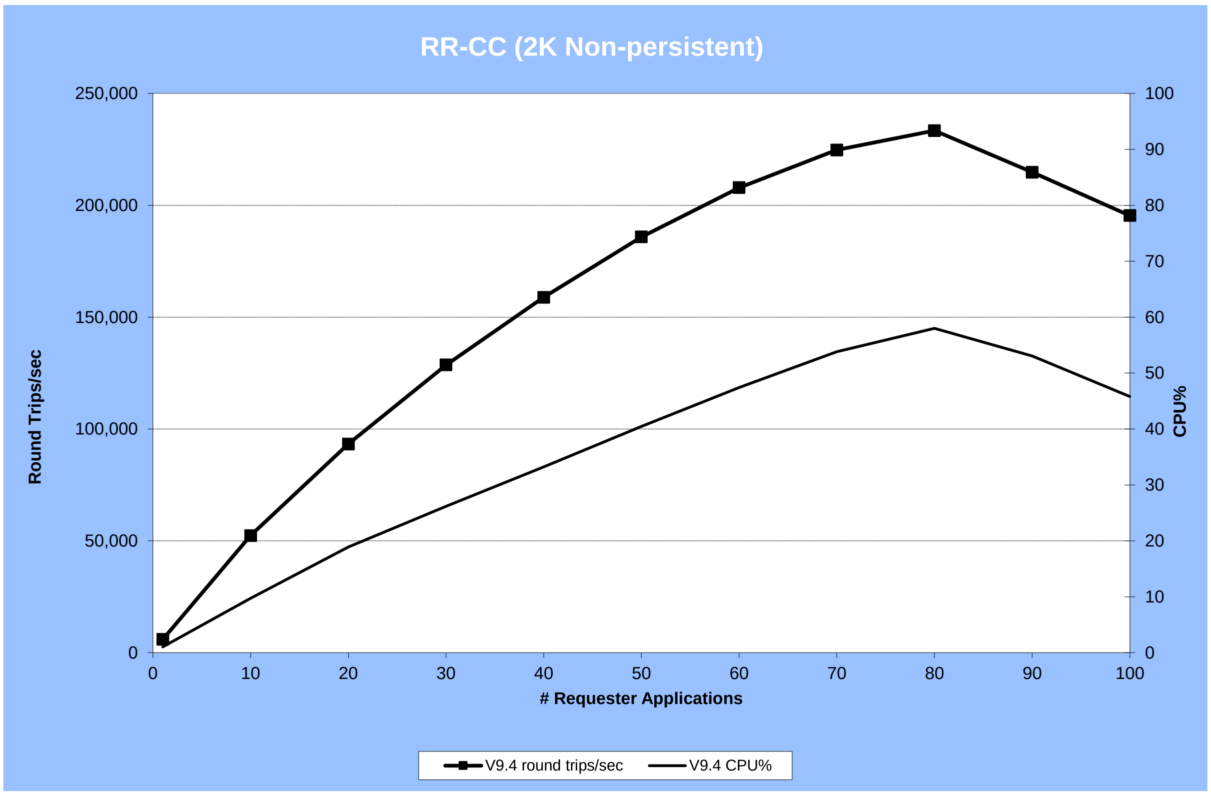

Figure 5 - Performance results for RR-CC (2KB non-persistent)

The test peaked at approximately 230,000 round trips/sec; in this instance the full CPU of the MQ server was not utilized, as the CPU utilisation at the client host limited the performance.

Peak round trip rates for all message sizes tested can be seen in the table below. The 200KB and 2MB scenarios are being limited by the 100Gb network that links the client and server machines.

Test	V9.4		
	Max Rate*	CPU%	Clients
RR-CC (2K Non-persistent)	233,378	58.02	80
RR-CC (20K Non-persistent)	208,324	59.91	80
RR-CC (200K Non-persistent)	28,746	21	40
RR-CC (2MB Non-persistent)	2,820	42.24	100

*Round trips/sec

TABLE 3 - PEAK RATES FOR WORKLOAD RR-CC (NON-PERSISTENT)

6.1.1 Test setup

Workload type: RR-CC (see section 5.2)

Hardware: Server 1, Client 1 (see section A.1)

6.2 RR-DQ-BB Workload

The distributed queuing scenarios use workload type RR-DQ-BB (see section 5.3) where locally bound requesters put messages onto a remote queue.

The throughput will be sensitive to network tuning and server channel setup amongst other things. All the tests in this section utilise multiple send/receive channels. This particularly helps with smaller, non-persistent messages when the network is under-utilised.

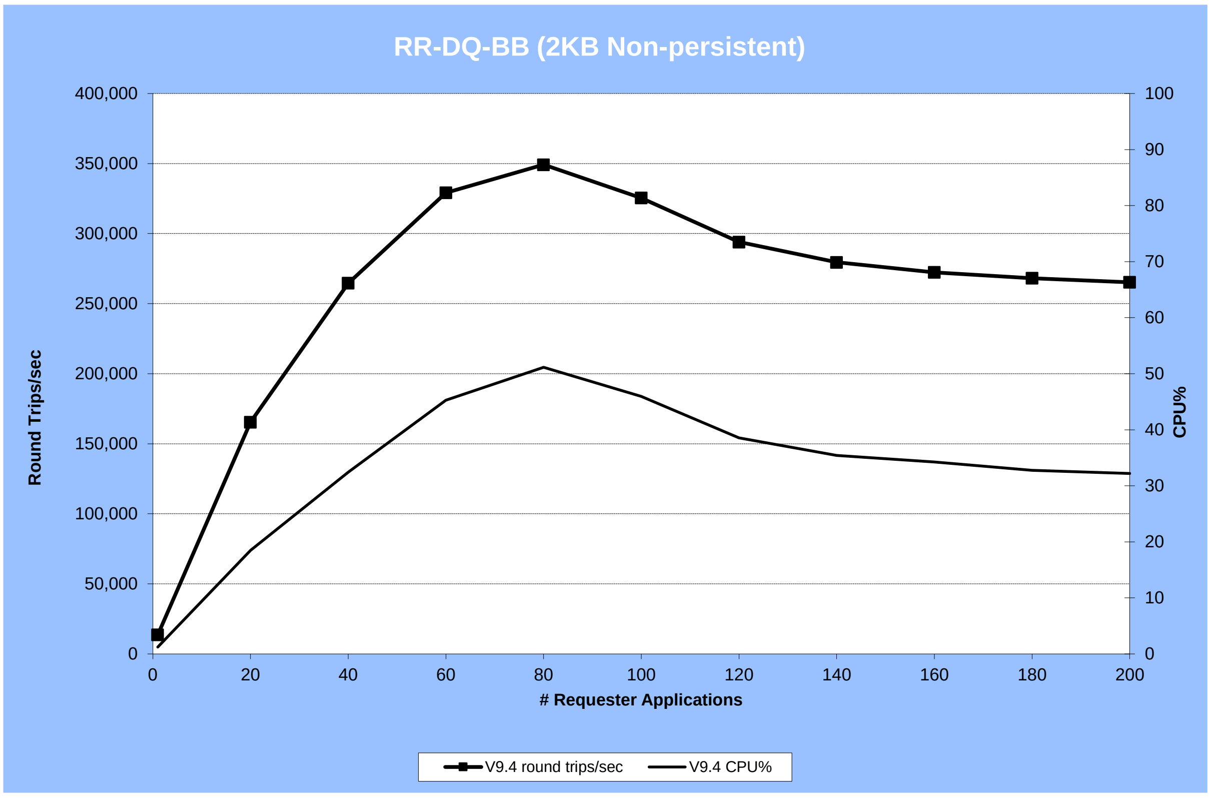

Figure 6 - Performance results for RR-DQ-BB (2KB non-persistent)

The distributed queuing test exhibits good scaling with CPU being the limiting factor at the client machine as the number of requester clients increases.

Peak round trip rates for all message sizes tested can be seen in the table below. The 200KB and 2MB measurements are again network limited by the 100Gb network. Note that these rates are higher than the RR-CC test in the previous section as the overall network traffic is lower per message (see the notes on network traffic in sections 0 and 5.3)

Test	V9.4		
	Max Rate*	CPU%	Clients
RR-DQ-BB (2KB Non-persistent)	349,192	51.14	80
RR-DQ-BB (20KB Non-persistent)	311,538	62.36	120
RR-DQ-BB (200KB Non-persistent)	57,563	24.68	40
RR-DQ-BB (2MB Non-persistent)	4,277	25.3	30

*Round trips/sec

TABLE 4 - FULL RESULTS FOR WORKLOAD RR-DQ-BB (NON-PERSISTENT)

6.2.1 Test setup

Workload type: RR-DQ-BB (see section 5.3)

Hardware: Server 1, Client 1 (see section A.1)

6.3 RR-CC JMS Workload

The test application is JMSPerfharness, which is run unrated (i.e. each requester sends a new message as soon as it receives the reply to the previous one). The JMS test is run with both requesters and responders in client mode on a remote host as JMSPerfharness is a relatively resource hungry application, utilising multiple JVMs to scale up the JMS connections.

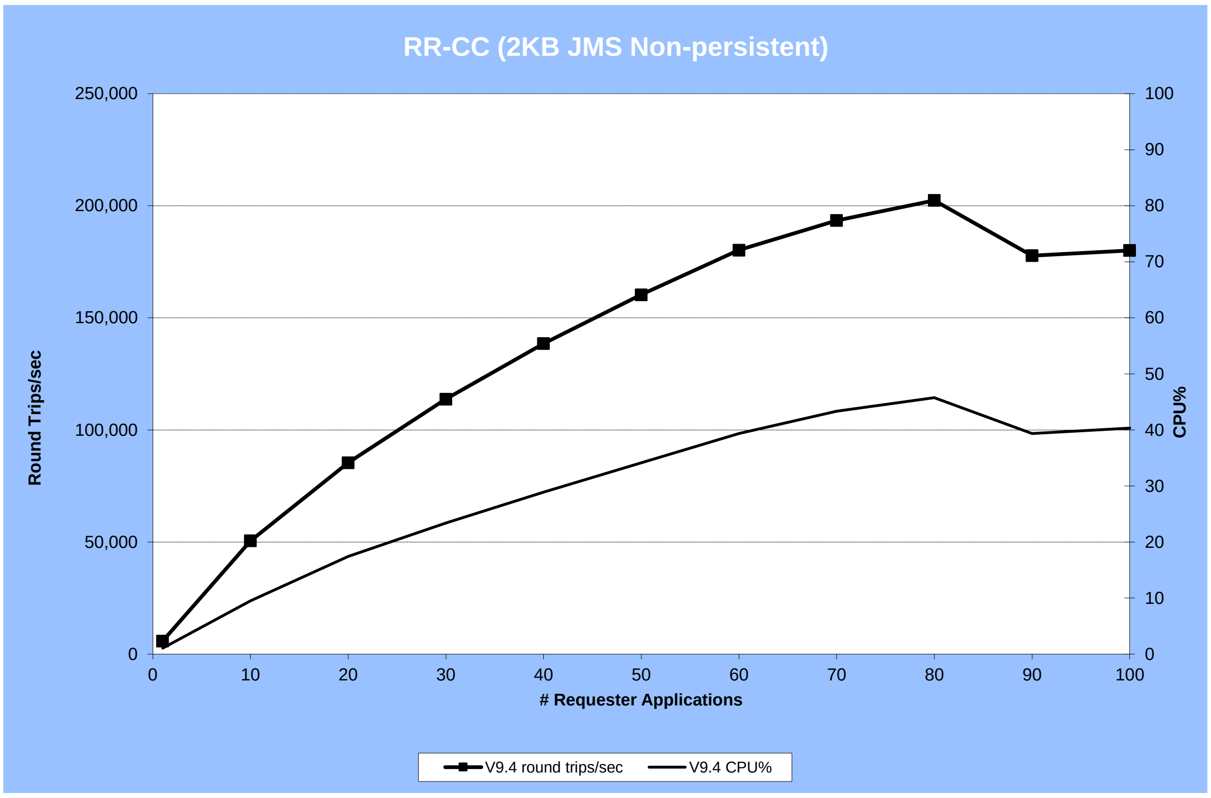

FIGURE 7 - PERFORMANCE RESULTS FOR RR-CC (2KB JMS NON-PERSISTENT)

Once again, the workload exhibits good scaling and peaking at approximately 200,000 round trips/sec, where the client host has utilized the available CPU.

Peak round trip rates for all message sizes tested can be seen in the table below. The 200KB and 2MB scenarios are network limited by the 100Gb network, whereas the 2K and 20K scenarios are limited by the CPU at the client.

Test	V9.4 Max Rate*	CPU%	Clients
RR-CC (2KB JMS Non-persistent)	202,379	45.74	80
RR-CC (20KB JMS Non-persistent)	169,132	42.84	70
RR-CC (200KB JMS Non-persistent)	24,316	26.48	160
RR-CC (2MB JMS Non-persistent)	2,170	27.67	160

*Round trips/sec

TABLE 5 - PEAK RATES FOR JMS (NON-PERSISTENT)

6.3.1 Test setup

Workload type: RR-CC (see section 5.2).

Message protocol: JMS

Hardware: Server 1, Client 1 (see section A.1)

6.4 RR-CC Workload with TLS

To illustrate the overhead of enabling TLS, results are provided comparing the performance of the 4 strongest TLS1.2 MQ CipherSpecs, with the baseline client bindings 2KB test presented in section 6.1.

The TLS 1.2 ciphers under test are shown below (all utilise 256bit encryption, and are FIPS compliant):

CipherSpec	**SuiteB**
ECDHE_ECDSA_AES_256_CBC_SHA384	No
ECDHE_ECDSA_AES_256_GCM_SHA384	Yes
ECDHE_RSA_AES_256_CBC_SHA384	No
ECDHE_RSA_AES_256_GCM_SHA384	No

Results for the suite B compliant CipherSpec (ECDHE_ECDSA_AES_256_GCM_SHA384) are plotted below. This cipherspec uses a GCM (Galois/Counter Mode) symmetric cipher. Performance testing showed that all GCM based CipherSpecs exhibited similar performance. CipherSpecs utilising the older CBC (Chain Block Cipher) symmetric cipher also exhibited similar performance to each other. One CBC CipherSpec (ECDHE_ECDSA_AES_256_CBC_SHA384) and a TLS 1.3 CipherSpec (TLS_AES_128_CCM_8_SHA256) are plotted below, for comparison.

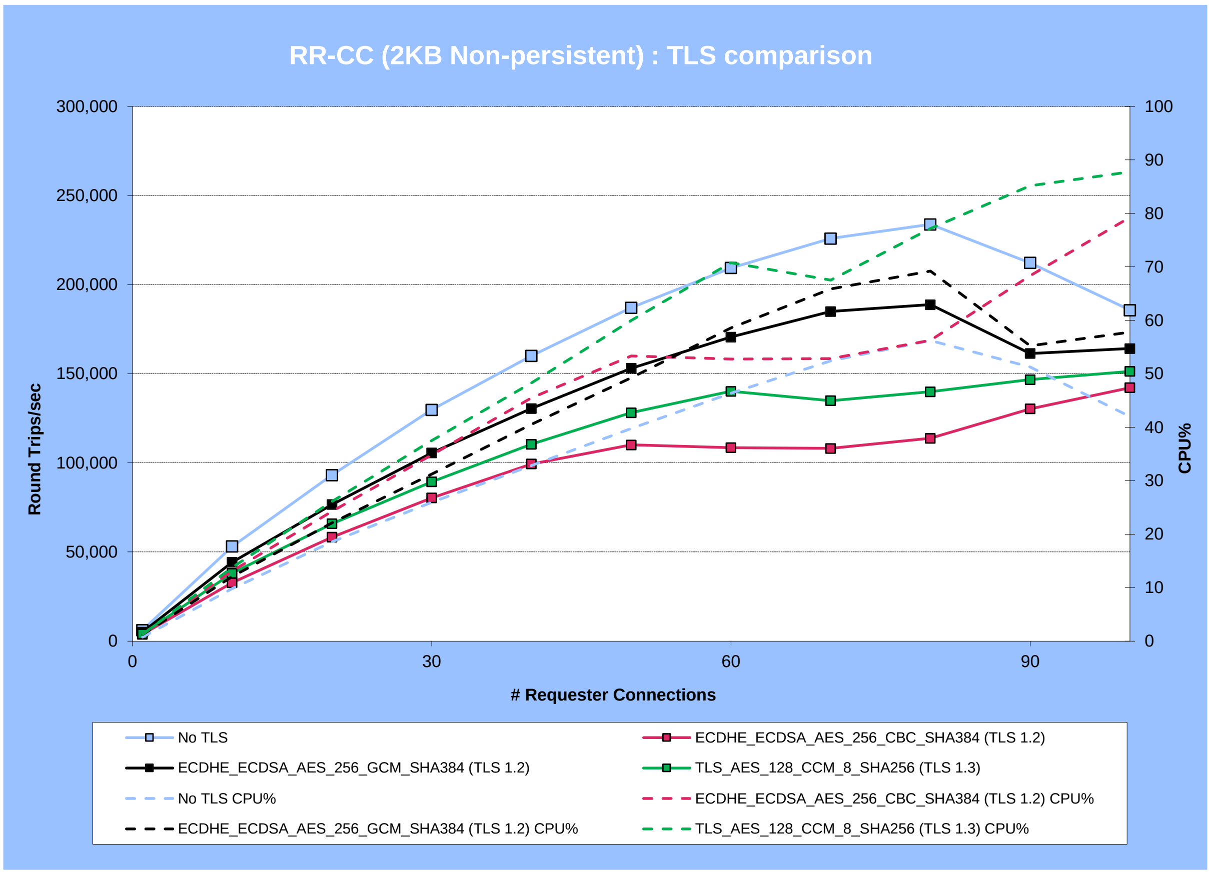

FIGURE 8 - PERFORMANCE RESULTS FOR RR-CC WITH TLS

All tests exhibited good scaling up to 100% of the CPU of the client machine. Throughput for TLS 1.2 GCM based CipherSpecs ran at approximately 80% of the throughput of a non-encrypted workload. TLS 1.2 CBC based CipherSpecs exhibited a greater overhead, running at approximately 61% of a non-encrypted workload.

All TLS 1.3 CipherSpecs exhibited a performance profile similar to TLS_AES_128_CCM_8_SHA256 in the plot above. (~65% of non encrypted workload)

Table 6 shows the peak rates achieved for the TLS 1.2 CipherSpecs tested, demonstrating the patterns of performance, depending on whether the symmetric key algorithm is CBC, or GCM based.

TLS 1.2 CipherSpec	V9.4 GM		
	Max Rate*	CPU%	Clients
No TLS	233,835	56	80
ECDHE_ECDSA_AES_256_CBC_SHA384	142,188	79	100
ECDHE_ECDSA_AES_256_GCM_SHA384	188,819	69	80
ECDHE_RSA_AES_256_CBC_SHA384	142,254	79	100
ECDHE_RSA_AES_256_GCM_SHA384	196,867	75	80

*Round trips/sec

TABLE 6 - PEAK RATES FOR MQI CLIENT BINDINGS (2KB NON-PERSISTENT) – TLS 1.2

Table 7 shows the peak rates achieved for all TLS 1.3 CipherSpecs.

TLS 1.3 CipherSpec	V9.4 GM Max Rate*	CPU%	Clients
No TLS	233,835	56	80
TLS_AES_128_CCM_8_SHA256	151,369	88	100
TLS_AES_256_GCM_SHA384	163,773	84	100
TLS_CHACHA20_POLY1305_SHA256	150,688	79	100
TLS_AES_128_GCM_SHA256	159,300	89	100
TLS_AES_128_CCM_SHA256	151,147	86	100

*Round trips/sec

TABLE 7 - PEAK RATES FOR MQI CLIENT BINDINGS (2KB NON-PERSISTENT) – TLS 1.3

6.4.1 Test setup

Workload type: RR-CC (see section 5.2)

Hardware: Server 1, Client 1 (see section A.1)

7 Persistent performance test results

The performance of persistent messaging is largely dictated by the capabilities of the underlying filesystem hosting the queue files, and more critically, the transaction log files. IBM MQ is designed to maximise throughput, regardless of the technology used, by aggregating writes where possible, to the transaction log, where they need to be synchronous to ensure transactional integrity.

The performance of persistent messaging is therefore dependant on the machine hosting MQ, *and* the I/O infrastructure. Some comparisons are shown below between non-persistent and persistent messaging for local storage.

7.1 RR-CC Workload

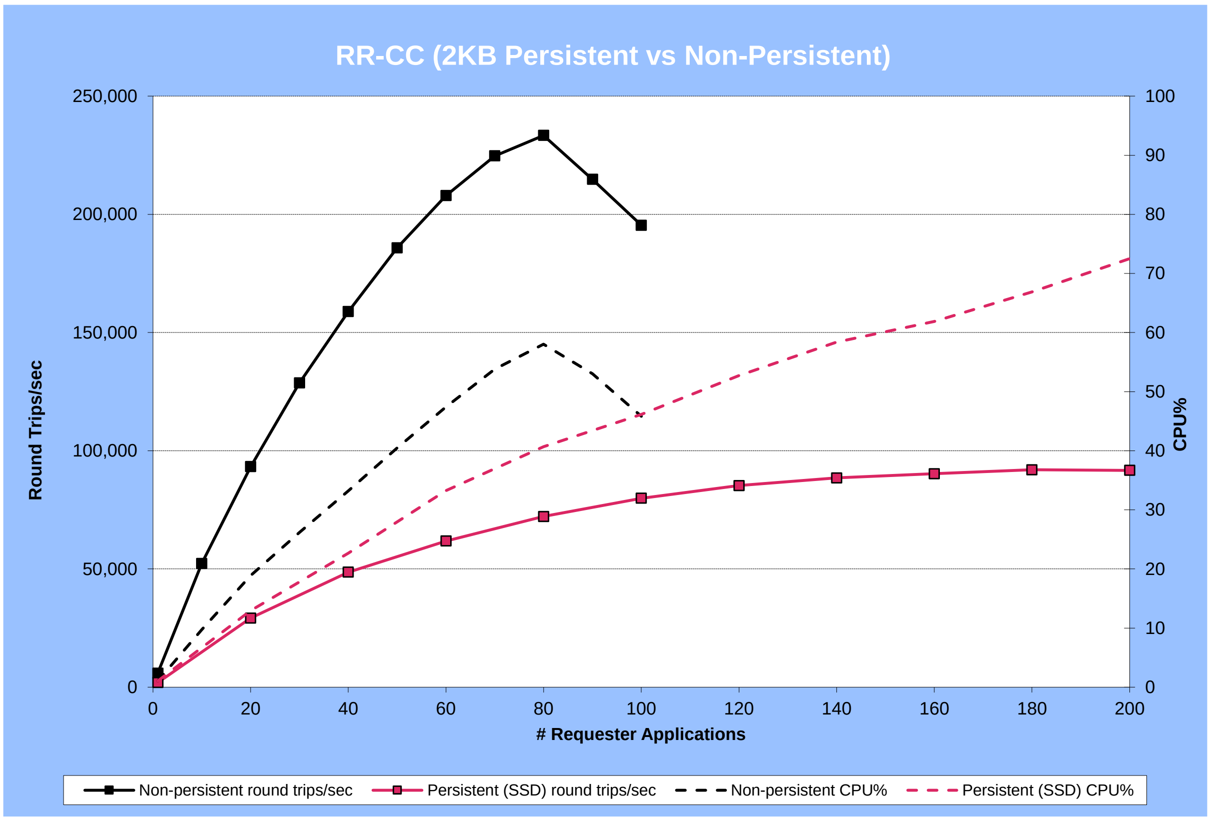

FIGURE 9 - PERFORMANCE RESULTS FOR RR-CC (2KB NON-PERSISTENT VS PERSISTENT)

Figure 9 shows the results from running the RR-CC workload with 2KB non-persistent and persistent messages.

Note that for smaller message sizes (as for 2KB, above), higher rates of throughput in persistent scenarios are attained when there is a greater deal of concurrency (i.e. requester applications) as this enables the logger component of IBM MQ to aggregate log data into larger, more efficient writes to disk.

Non-persistent workloads are typically limited by the CPU, whilst the transaction log I/O is the limiting factor for the persistent workloads. As the message size goes up, the time spent on the transaction log write becomes a larger factor. The level of concurrency needed to reach the limitations of the filesystem also drops as the message size increases.

In these tests, the machines are connected via 100Gb links in the same data centre. With network links that are higher latency or lower bandwidth, the difference between nonpersistent and persistent throughput will be less, as the network becomes a significant part of the bottleneck. Peak round trip rates for all message sizes tested, for persistent & non-persistent scenarios can be seen in Table 8 & Table 9 below.

Test	V9.4		
	Max Rate*	CPU%	Clients
RR-CC (2K Non-persistent)	233,378	58.02	80
RR-CC (20K Non-persistent)	208,324	59.91	80
RR-CC (200K Non-persistent)	28,746	21	40
RR-CC (2MB Non-persistent)	2,820	42.24	100

*Round trips/sec

TABLE 8 - PEAK RATES FOR WORKLOAD RR-CC (NON-PERSISTENT)

Test	V9.4		
	Max Rate*	CPU%	Clients
RR-CC (2KB Persistent)	92,633	85.66	300
RR-CC (20KB Persistent)	77,795	79.25	300
RR-CC (200KB Persistent)	17,539	42.18	140
RR-CC (2MB Persistent)	1,526	22.3	40

*Round trips/sec

TABLE 9 - PEAK RATES FOR WORKLOAD RR-CC (PERSISTENT)

7.1.1 Test setup

Workload type: RR-CC (see section 5.2)

Hardware: Server 1, Client 1 (see section A.1)

Appendix A: Test configurations

A.1 Hardware/Software – Set 1

All the testing in this document (apart from when testing results are shown from a different platform and are clearly identified as such) was performed on the following hardware and software configuration:

A.1.1 Hardware

Server1 and Client1 are two identical machines:

Model: Lenovo ThinkSystem SR630 V3 – [7D73]

CPU: 2 x 16 cores; Intel® Xeon® 6544Y @ 3.60GHz

RAM: 256GB RAM

Disk: 2x1.6TB NVMe SSD in RAID-0

RAID: Intel® VRoc

Network: 100Gb Ethernet located on isolated performance LAN

A.1.2 Software

Microsoft Windows Server 2022 Datacenter

JMSPerfHarness test driver (see Appendix C:)

MQ-CPH MQI test driver (see Appendix C:)

IBM MQ V9.4

A.2 IBM MQ Configuration

The following parameters are added or modified in the qm.ini files for the tests run in this report:

```
Channels:
   MQIBindType=FASTPATH
   MaxActiveChannels=5000
   MaxChannels=5000
Log:
   LogBufferPages=4096
   LogFilePages=16384
   LogPrimaryFiles=16
   LogSecondaryFiles=2
   LogType=CIRCULAR
   LogWriteIntegrity=TripleWrite
TuningParameters:
   DefaultPQBufferSize=10485760
   DefaultQBufferSize=10485760
```

For large message sizes (20K, 200K & 2MB), the queue buffers were increased further to:

```
DefaultPQBufferSize=104857600
DefaultQBufferSize=104857600
```

For message sizes (200K and above), the number of LogPrimaryFiles was increased to 64.

Note that large queue buffers may not be needed in your configuration. Writes to the queue files are asynchronous, taking advantage of OS buffering. Large buffers were set in the runs here, as a precaution only.

Appendix B: Glossary of terms used in this report

CD	Continuous delivery
JMSPerfharness	JMS based, performance test application (https://github.com/ot4i/perf-harness)
LTS	Long term service
MQ-CPH	C based, performance test application (https://github.com/ibm-messaging/mq-cph)

Appendix C: Resources

MQ Performance GitHub Site
https://ibm-messaging.github.io/mqperf/

Linear Logging Improvements Blog Article
https://www.ibm.com/developerworks/community/blogs/messaging/entry/Logger_enhancements_for_MQ_v9_0_2?lang=en

Implicit Syncpoint Blog Article
https://developer.ibm.com/messaging/2018/04/24/implicit-syncpointing-persistent-messages-put-outside-syncpoint

MQ-CPH (The IBM MQ C Performance Harness)
https://github.com/ibm-messaging/mq-cph

JMSPerfHarness
https://github.com/ot4i/perf-harness

Persistent Messaging Performance Paper
https://ibm-messaging.github.io/mqperf/mqio_v1.pdf

Transaction Log Sizing Documentation
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q018470_.htm

Best Practices Performance paper

https://ibm-messaging.github.io/mqperf/MQ_Performance_Best_Practices_v1.0.1.pdf