IBM MQ V9.4 for Linux (x86-64 platform) Performance Report

Version 1.0 - September 2024

Paul Harris

IBM MQ Performance

IBM UK Laboratories

Hursley Park

Winchester

Hampshire

UK

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, September 2024.

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

DISCLAIMERS

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

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

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

WARRANTY AND LIABILITY EXCLUSION

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

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

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

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

ERRORS AND OMISSIONS

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

INTENDED AUDIENCE

This report is intended for architects, systems programmers, analysts and programmers wanting to understand the performance characteristics of 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.

LOCAL AVAILABILITY

References in this report to IBM products or programs do not imply that IBM intends to make these available in all countries in which IBM operates. Consult your local IBM representative for information on the products and services currently available in your area.

ALTERNATIVE PRODUCTS AND SERVICES

Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

USE OF INFORMATION PROVIDED BY YOU

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

TRADEMARKS AND SERVICE MARKS

The following terms used in this publication are trademarks of their respective companies in the United States, other countries or both:

- **IBM Corporation:** IBM
- **Oracle Corporation:** Java

Other company, product, and service names may be trademarks or service marks of others.

EXPORT REGULATIONS

You agree to comply with all applicable export and import laws and regulations.

Preface

Target audience

The report is designed for people who:

- Will be designing and implementing solutions using IBM MQ v9.4 for Linux on x86_64.
- Want to understand the performance limits of IBM MQ v9.4 for Linux on x86_64.
- Want to understand what actions may be taken to tune IBM MQ v9.4 for Linux on x86_64.

The reader should have a general awareness of the Linux operating system and of IBM MQ 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. A separate paper on general performance best practises has been made available here:

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

Contents

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.

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 paul_harris@uk.ibm.com.

Contents

TABLES

FIGURES

1 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 the V9.3 release of IBM MQ.

The hardware used in the report is identical to that used in the MQ V9.3 LTS performance report, but the version of RedHat Linux is newer and all security patching between the release of MQ V9.3 and MQ 9.4 has been applied. As such it is possible that some results may be seen to be slightly lower than those published in the V9.3 report since security patching will tend to add pathlength in some cases.

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.

2 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

2.1 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).

```
export AMQ_IODELAY_INMS=YES
export AMQ_IODELAY=7000
```

Using these values for a test where the recovery log is hosted on nfs, we can introduce a latency of 10ms on the link to the nfs server (‘en1’ in this case):

```
tc qdisc add dev en1 root netem delay 10000us
```

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.

2.2 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.

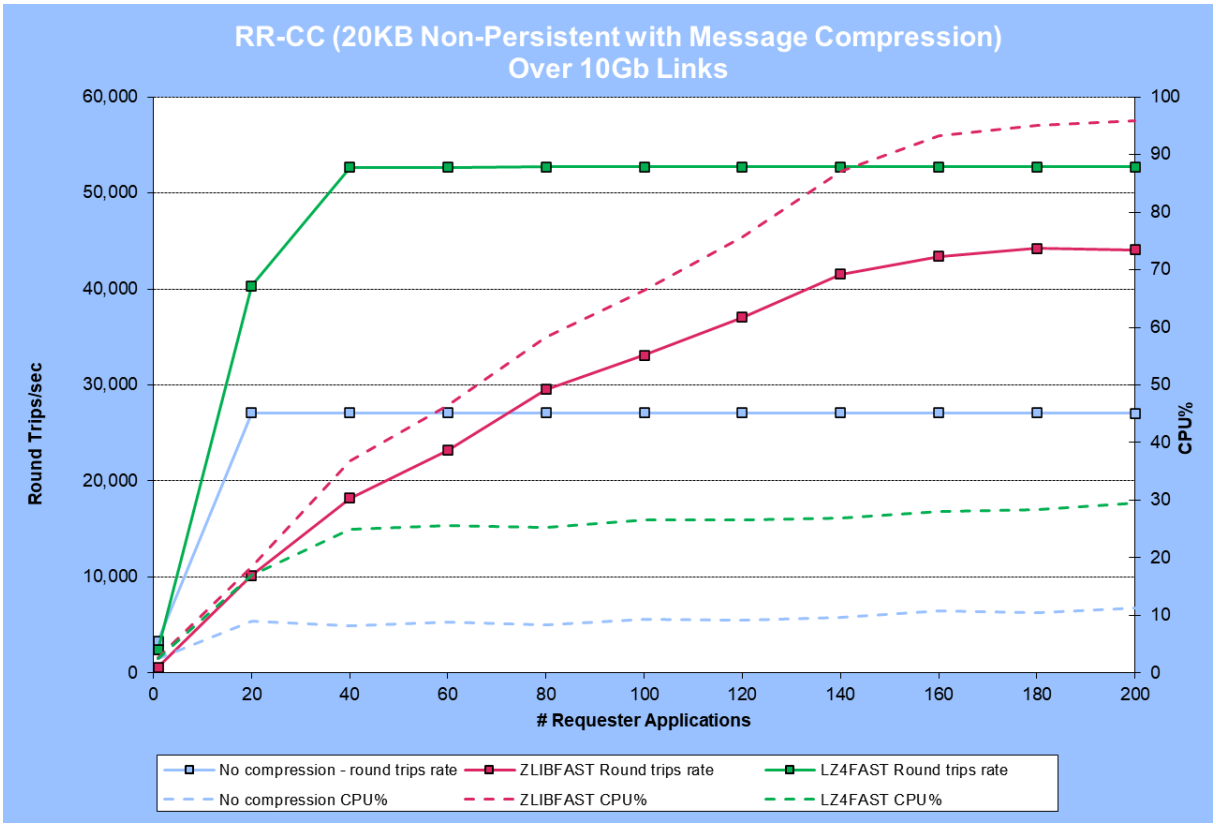

FIGURE 1- EFFECT OF MESSAGE COMPRESSION OVER NARROW BANDWIDTH NETWORK CONNECTION (10GB ETHERNET)

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).

For the 10Gb linked environment for the 20K results above, varying degrees of benefit were recorded when using compression.

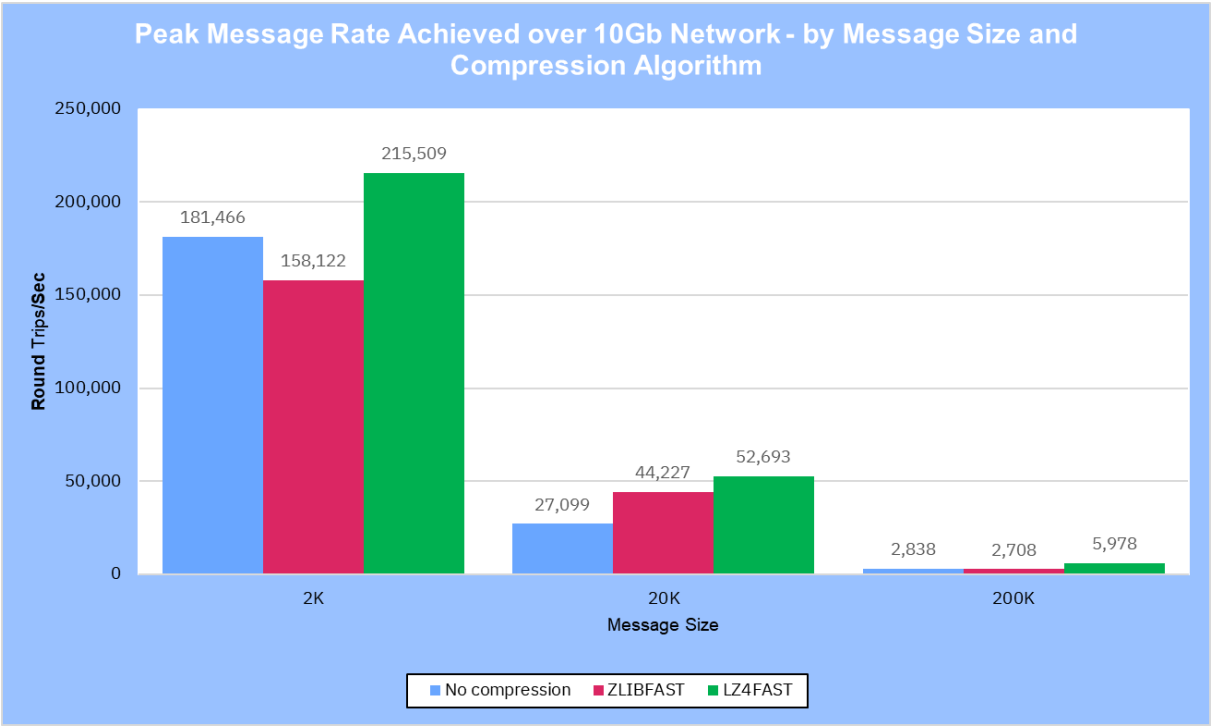

FIGURE 2- PEAK MESSAGE RATES BY MESSAGE SIZE AND COMPRESSION ALGORITHM.

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 only faster for 20K messages.

For the 200K message test, a maximum of 300 requesters were started and whilst the no-compression and LZ4FAST variants of the test had peaked, the ZLIBHIGH test was still climbing as more applications were started (so it's likely that with an even higher number of applications and given spare CPU capacity, it would have achieved a higher message rate than the uncompressed test.

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.

Msg Size	Compression Rate			
	ZLIBFAST	LZ4FAST	ZLIBHIGH	LZ4HIGH
2KB	34%	16%	37%	19%
20KB	56%	43%	58%	47%
200KB	59%	45%	61%	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.

2.2.1 Test setup.

Workload type: RR-CC (see section 3.1).

Hardware: Server 1, Client 1, Client 2 (see section A.1).

3 Base MQ Performance Workloads

Table 2 (below) lists the workloads used in the generation of performance data for base MQ (that is standard messaging function) in 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-DQ-BB	Distributed queueing between two queue managers on separate hosts, with binding mode requesters and responders.
RR-CC	Client mode requesters, and responders on separate, unique hosts

TABLE 2 - WORKLOAD TYPES

Binding mode connections use standard MQ bindings. Client mode connections use fastpath channels and listeners (trusted) and have SHARECNV set to 1, which is the recommended value for performance.

RR-CB & RR-DQ-BB are described in the following section. The remaining two workloads differ only in the location of the MQ applications, which is made clear in the results presented in this report.

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.

3.1 RR-CC Workload

(Client mode requesters with client mode responders.)

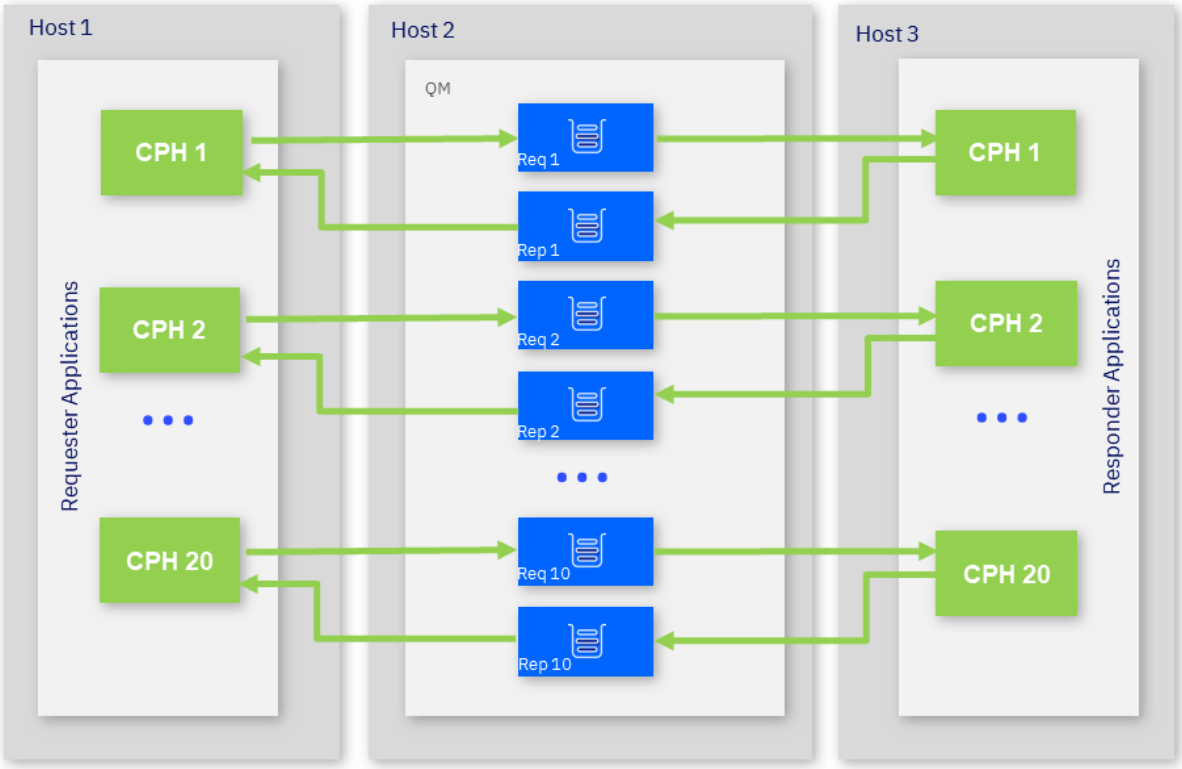

FIGURE 3 - RR-CC TOPOLOGY

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. Additional machines may be used to drive the requester applications where necessary.

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. 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:

- 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.
- 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.

- 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 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.

Network Flows:

As the topology utilises separate hosts 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.

3.2 RR-DQ-BB Workload

(Distributed queueing between two queue managers on separate hosts, with binding mode requesters and responders).

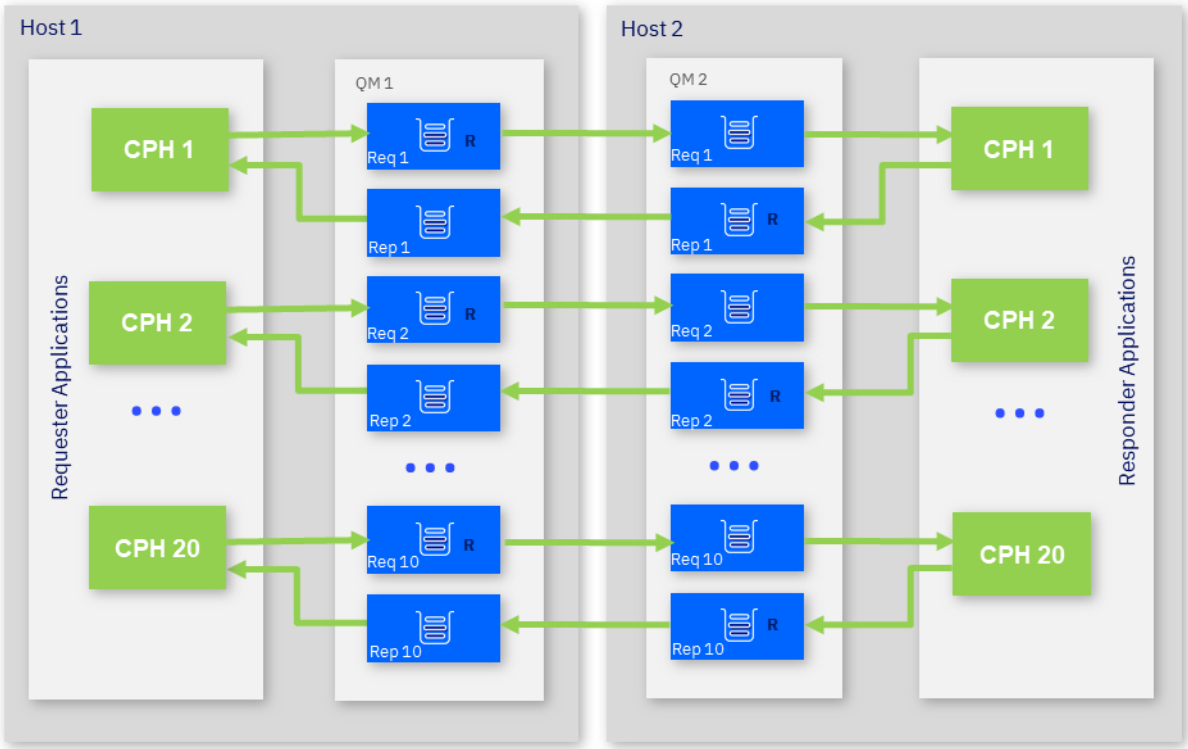

FIGURE 4 - REQUESTER-RESPONDER WITH REMOTE QUEUE MANAGER (REMOTE RESPONDERS).

This is a distributed queuing version of the requester-responder topology detailed in section 3.2. All *MQPUT*s are to remote queues (marked with 'R' in the diagram above), so 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.

Network Flows:

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.

4 Non-Persistent Performance Test Results

Full performance test results are detailed below. The test results are presented by broad categories 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).

4.1 RR-CC Workload

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

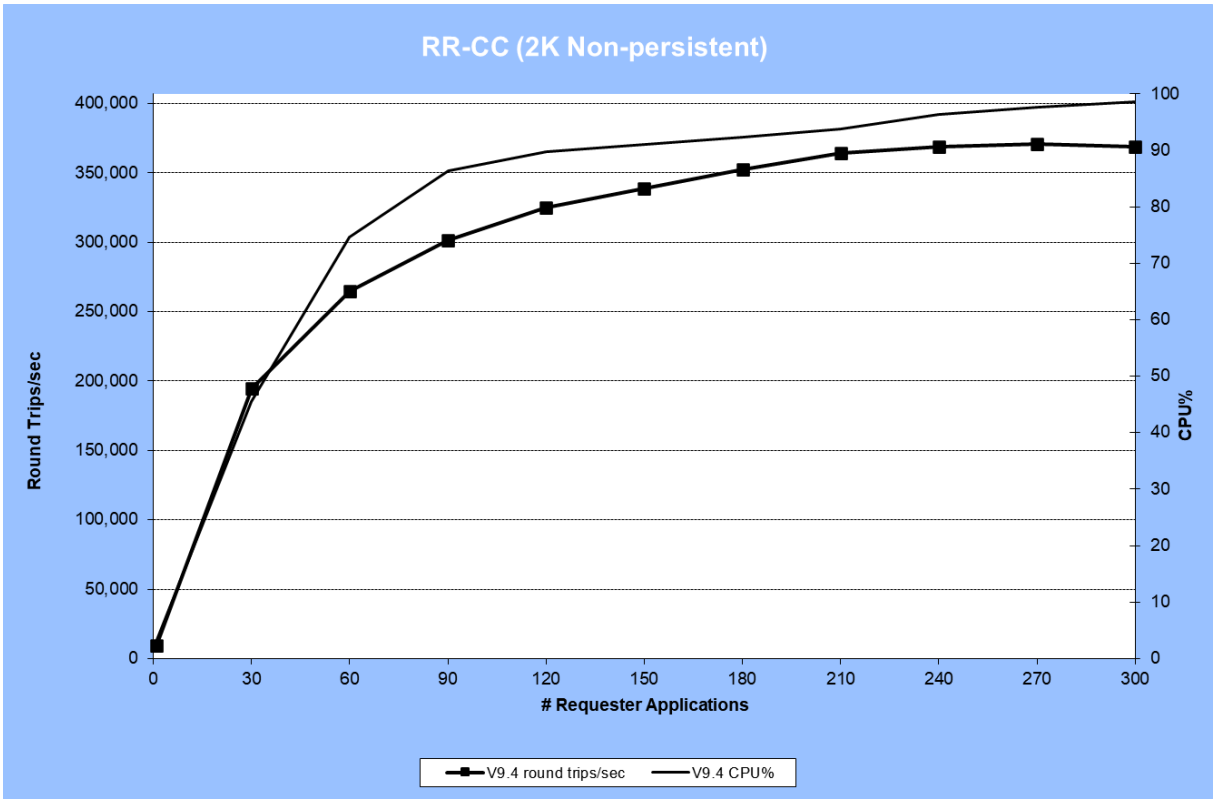

FIGURE 5 - PERFORMANCE RESULTS FOR RR-CC (2KB NON-PERSISTENT)

The test peaked at approximately 371,000 round trips/sec, approaching full CPU utilisation of the MQ server.

Peak round trip rates for all message sizes tested can be seen in the table below. The 200KB and 2MB scenarios are approaching saturation of the 100Gb network links between the client and server machines (hence lower CPU% for the highest message rate).

Test	V9.4 Max Rate*	CPU%	Clients
RR-CC (2K Non-persistent)	370,602	97.69	270
RR-CC (20K Non-persistent)	261,852	92.66	200
RR-CC (200K Non-persistent)	27,699	32.43	50
RR-CC (2MB Non-persistent)	2,613	32.61	40

Round trips/sec

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

4.1.1 Test setup.

Workload type: RR-CC (see section 3.1).

Hardware: Server 1, Client 1, Client 2 (see section A.1).

4.2 RR-DQ-BB Workload

(Distributed queueing between two queue managers on separate hosts, with binding mode requesters and responders).

The distributed queuing scenarios use workload type RR-DQ-BB (see section 3.2) 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.

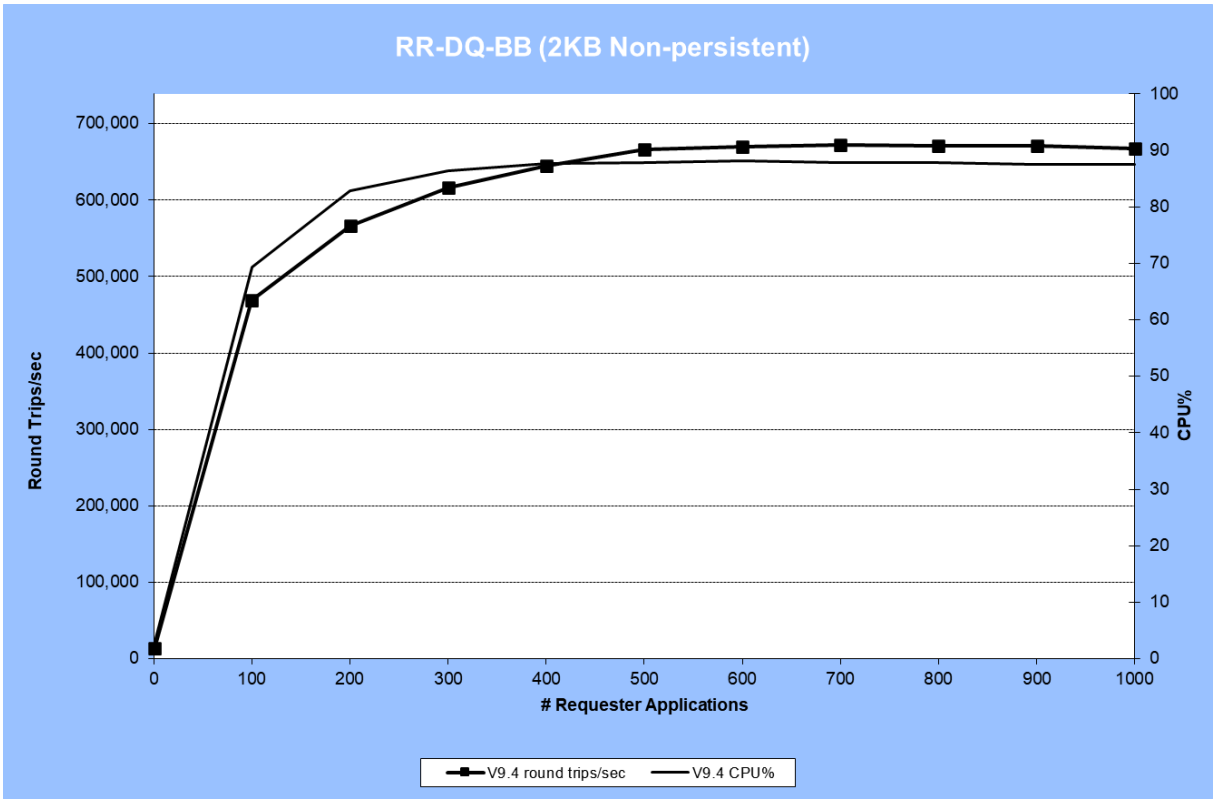

FIGURE 6 - PERFORMANCE RESULTS FOR RR-DQ-BB (2KB NON-PERSISTENT)

The distributed queuing test exhibits good scaling with CPU being the limiting factor as the number of 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 links between the hosts. 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 3.1 and 3.2)

Test	V9.4 Max Rate*	CPU%	Clients
RR-DQ-BB (2KB Non-persistent)	672,243	87.84	700
RR-DQ-BB (20KB Non-persistent)	354,563	65.32	240
RR-DQ-BB (200KB Non-persistent)	56,883	32.18	50
RR-DQ-BB (2MB Non-persistent)	4,609	36.57	45

Round trips/sec

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

4.2.1 Test setup.

Workload type: RR-DQ-BB (see section 3.2).

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

4.3 RR-CC JMS Workload

This 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 remote hosts as JMSPerfharness is a relatively resource hungry application, utilising multiple JVMs to scale up the JMS connections.

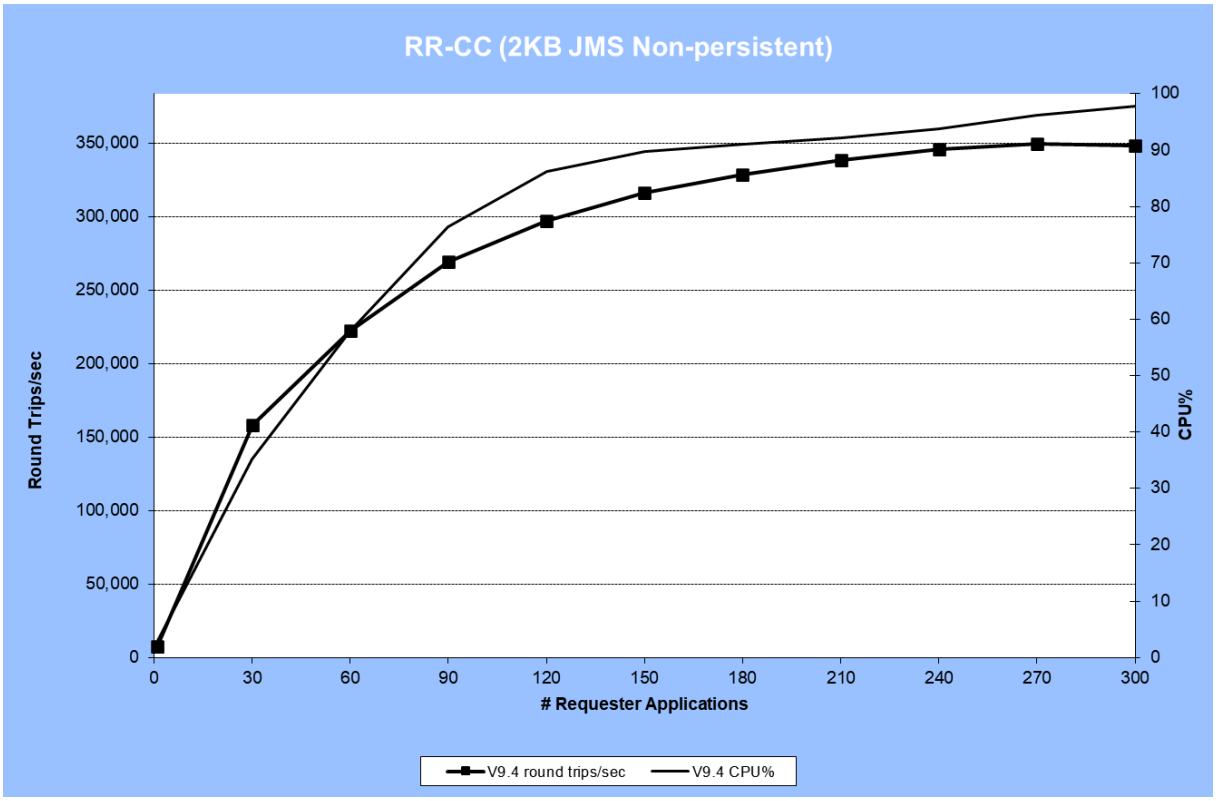

FIGURE 7 - PERFORMANCE RESULTS FOR RR-CC (2KB JMS NON-PERSISTENT)

Once again, the workload exhibits good scaling up to 100% of the CPU (the limiting factor), peaking at approximately 350,000 round trips/sec.

Peak round trip rates for all message sizes tested can be seen in the table below.

Test	V9.4 Max Rate*	CPU%	Clients
RR-CC (2KB JMS Non-persistent)	349,939	96.13	270
RR-CC (20KB JMS Non-persistent)	239,284	93.73	240
RR-CC (200KB JMS Non-persistent)	27,567	61.59	200
RR-CC (2MB JMS Non-persistent)	2,653	62.35	180

***Round trips/sec**

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

4.3.1 Test setup.

Workload type: RR-CC (see section 3.1).

Message protocol: JMS.

Hardware: Server 1, Client 1, Client 2 (see section A.1).

5 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 MQ recovery log files. Writes to the recovery log need to be synchronous to ensure transactional integrity, but IBM MQ is designed to maximise throughput, by aggregating writes where possible. Aggregation of log writes is dependent on a concurrent workload (i.e. multiple applications connected and committing data to the queue manager concurrently, such that the MQ logger component can aggregate data into larger, more efficient file writes and mitigate the higher latency of some file systems).

The performance of persistent messaging is therefore dependant on the machine hosting MQ, the degree of concurrency, *and* the I/O infrastructure. Some comparisons are shown below between non-persistent and persistent messaging for local storage, and then results for V9.4 in a separate environment (x64 Linux with SAN, SSD & NFS filesystems) are shown to demonstrate the impact of recovery log location.

5.1 RR-CC Workload

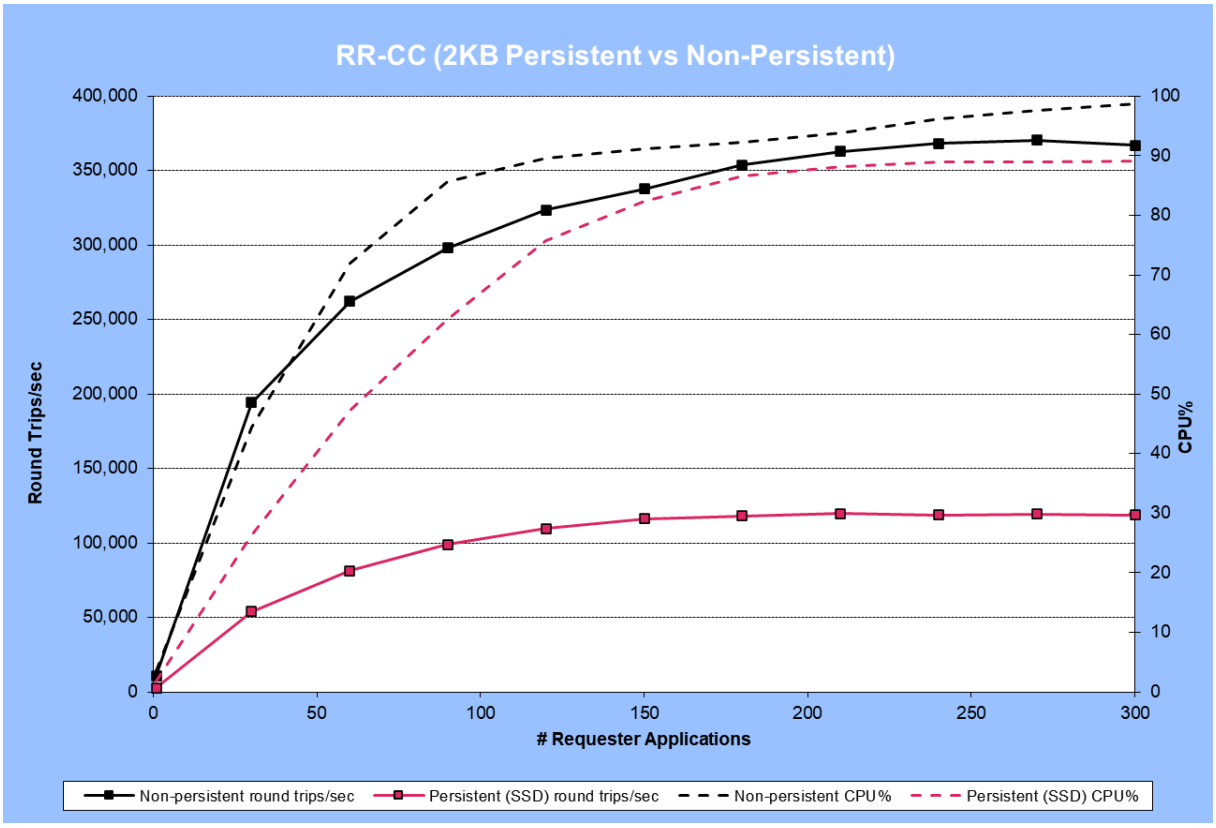

FIGURE 8 - PERFORMANCE RESULTS FOR RR-CC (2KB NON-PERSISTENT VS PERSISTENT)

Figure 8 shows results from running the RR-CC workload with 2KB non-persistent and persistent messages, on the same server used for the non-persistent scenarios in the previous sections.

The non-persistent workload reaches an optimal value at 270 requesters where, the CPU approaches 100% utilisation. Adding more requesters degrades performance, increasing context switching on an already saturated server.

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 to perform much larger writes (as described above).

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 non-persistent 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 6 & Table 7 below.

Test	V9.4 Max Rate*	CPU%	Clients
RR-CC (2K Non-persistent)	370,602	97.69	270
RR-CC (20K Non-persistent)	261,852	92.66	200
RR-CC (200K Non-persistent)	27,699	32.43	50
RR-CC (2MB Non-persistent)	2,613	32.61	40

***ROUND TRIPS/ SEC**

TABLE 6 - PEAK RATES FOR WORKLOAD RR-CC (NON-PERSISTENT)

Test	V9.4 Max Rate*	CPU%	Clients
RR-CC (2KB Persistent)	119,846	88.2	210
RR-CC (20KB Persistent)	91,968	82.81	225
RR-CC (200KB Persistent)	14,993	31.63	100
RR-CC (2MB Persistent)	1,641	25.95	45

***ROUND TRIPS/ SEC**

TABLE 7 - PEAK RATES FOR WORKLOAD RR-CC (PERSISTENT)

The non-persistent numbers are for comparison with persistent messaging, to illustrate what the impact of logging can be.

The recovery log I/O is the limiting factor for the persistent workloads here, as expected. As the message size goes up, the time spent on the recovery log write becomes a larger factor, so although the bytes per sec is more, the overall CPU utilisation is lower. The level of concurrency needed to reach the limitations of the filesystem also drops as the message size increases.

5.1.1 Test setup.

Workload type: RR-CC (see section 3.1).

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

5.2 Impact of Different File Systems on Persistent Messaging Performance

A separate paper has been published, with illustrative results, for SSD, SAN and NFS hosted filesystems, along with some guidance, on best practises, and monitoring.

https://ibm-messaging.github.io/mqperf/mqio_v1.pdf

If possible, you should assess the performance of a new application, with non-persistent messaging first. If the target rate of messaging is met, then calculate the required bandwidth of the filesystem hosting the recovery logs.

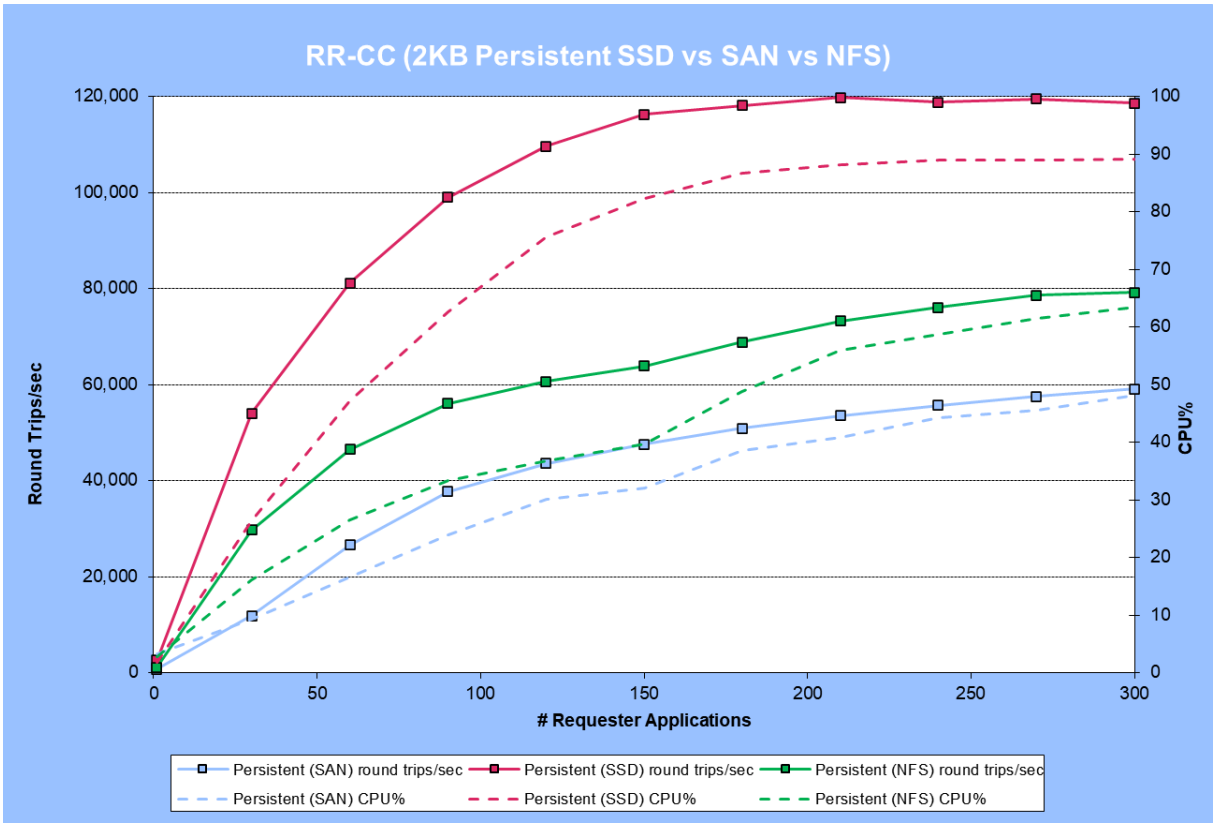

FIGURE 9 - PERFORMANCE RESULTS FOR RR-CC PERSISTENT MESSAGING LOGGING TO SSD, SAN & NFS

To illustrate the impact that the filesystem hosting the recovery logs can have, Figure 9 shows results from running the RR-CC workload with persistent messaging where the recovery logs are on a local SSD or hosted remotely (SAN or NFS).

As expected, logging to a local SSD is a lot faster. The SAN tests are limited by the bandwidth of the SAN switch (16Gb ports). For NFS, the network link is 100Gb, but

independent tests showed a limit of around 27Gb/s for single threaded transfers (which the MQ logger must by design perform, to maintain data integrity). The MQ logger will perform larger writes as the number of applications increase but there is a 1MB write size limit for NFS, in the Linux kernel.

Table 8 below, shows the peak rates achieved for each filesystem tested, across a range of message sizes.

Test	V9.4 Max Rate*	CPU%	Clients
RR-CC (2KB Persistent)	119,846	88.2	210
RR-CC (2KB Persistent - SAN)	59,124	48.17	300
RR-CC (2KB Persistent - NFS)	79,254	63.48	300
RR-CC (20KB Persistent)	91,968	82.81	225
RR-CC (20KB Persistent - SAN)	19,161	22.31	300
RR-CC (20KB Persistent - NFS)	22,694	24.9	300
RR-CC (200KB Persistent)	14,993	31.63	100
RR-CC (200KB Persistent - SAN)	2,839	9.29	120
RR-CC (200KB Persistent - NFS)	3,502	10.46	120
RR-CC (2MB Persistent)	1,641	25.95	45
RR-CC (2MB Persistent - SAN)	273	5.69	20
RR-CC (2MB Persistent - NFS)	343	6.74	40

Round trips/ sec

TABLE 8 - TABLE 9 - PEAK RATES FOR WORKLOAD RR-CC (PERSISTENT SSD VS SAN VS NFS)

1.1.1 Test setup.

Workload type: RR-CC (see section 3.1).

Hardware: Server 1, with client 1 machine acting as NFS server. (see section A.1).

6 RR-CC Workload with TLS

(Client mode requesters and responders on separate hosts).

To illustrate the overhead of enabling TLS to encrypt traffic between the client applications and the queue manager, results are shown below comparing the performance of the 4 strongest TLS1.2 MQ CipherSpecs, and all TLS1.3 MQ.

The following TLS 1.2 CipherSpecs were tested (all utilise 256bit encryption and are FIPS compliant).

- ECDHE_ECDSA_AES_256_CBC_SHA384
- ECDHE_ECDSA_AES_256_GCM_SHA384 (Suite B compliant)
- ECDHE_RSA_AES_256_CBC_SHA384
- ECDHE_RSA_AES_256_GCM_SHA384

Results for the suite B compliant CipherSpec (ECDHE_ECDSA_AES_256_GCM_SHA384), along with an older, CBC based CipherSpec (ECDHE_RSA_AES_256_CBC_SHA384) and a TLS 1.3 CipherSpec (TLS_AES_128_CCM_8_SHA256) are plotted below. As will be seen, the remaining tested CipherSpecs exhibited a performance profile similar to one of these plots.

6.1 TLS Non-Persistent Results

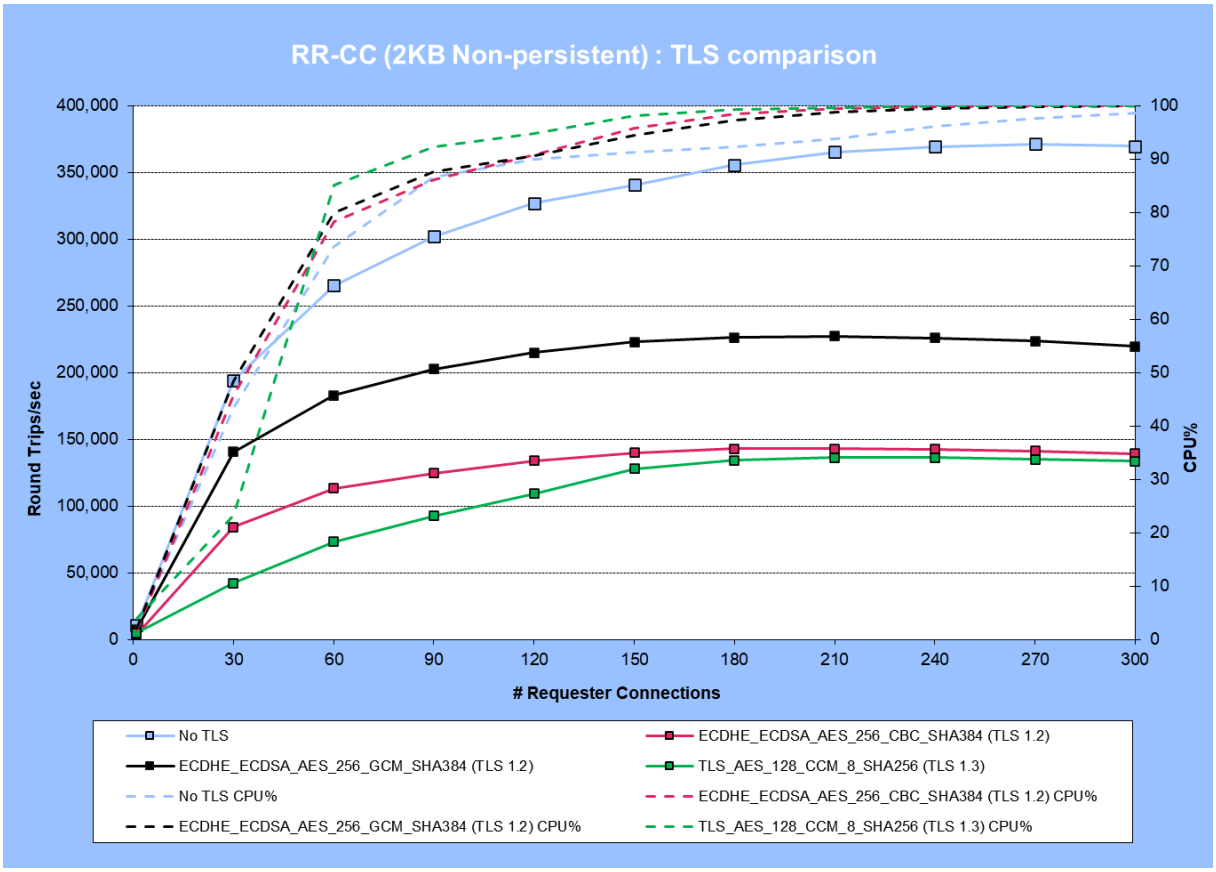

FIGURE 10 - PERFORMANCE RESULTS FOR RR-CC WITH TLS

The ECDHE_ECDSA_AES_256_GCM_SHA384 CipherSpec uses a GCM (Galois/Counter Mode) symmetric cipher. Performance testing showed that all TLS 1.2 GCM based CipherSpecs exhibited similar performance. All the TLS 1.2 CipherSpecs utilising the older CBC (Chain Block Cipher) symmetric cipher exhibited similar to ECDHE_ECDSA_AES_256_CBC_SHA384 in the plot above. All TLS 1.3 CipherSpecs exhibited a performance profile similar to TLS_AES_128_CCM_8_SHA256 in the plot above.

All tests exhibited scaling up to around 100% of the CPU of the machine. Throughput for GCM based CipherSpecs ran at approximately 61% of the throughput of a non-encrypted workload. CBC based CipherSpecs exhibited a greater overhead, running at approximately 38% of a non-encrypted workload. TLS 1.3 encryption is more expensive, achieving rates slightly below the TLS 1.2 CBC based CipherSpecs.

Table 9 shows the peak rates achieved for all 6 TLS 1.2 CipherSpecs tested, demonstrating the equivalence of performance, based on whether the symmetric key algorithm is CBC, or GCM based.

TLS 1.2 CipherSpec	V9.4 GM Max Rate*	CPU%	Clients
No TLS	371,209	98	270
ECDHE_ECDSA_AES_256_CBC_SHA384	142,715	99	210
ECDHE_ECDSA_AES_256_GCM_SHA384	226,220	99	210
ECDHE_RSA_AES_256_CBC_SHA384	143,004	99	210
ECDHE_RSA_AES_256_GCM_SHA384	227,269	99	210

Round trips/sec

TABLE 9 - PEAK RATES FOR MQI CLIENT BINDINGS (2KB NON-PERSISTENT) – TLS 1.2

Table 10 shows the peak rates achieved for all TLS 1.3 CipherSpecs.

TLS 1.3 CipherSpec	V9.4 GM Max Rate*	CPU%	Clients
No TLS	371,209	98	270
TLS_AES_128_CCM_8_SHA256	136,839	100	210
TLS_AES_256_GCM_SHA384	141,445	100	240
TLS_CHACHA20_POLY1305_SHA256	136,317	100	210
TLS_AES_128_GCM_SHA256	142,708	100	240
TLS_AES_128_CCM_SHA256	127,167	100	300

Round trips/sec

TABLE 10 - PEAK RATES FOR MQI CLIENT BINDINGS (2KB NON-PERSISTENT) – TLS 1.3

6.1.1 Test setup.

Workload type: RR-CC (see section 3.1).

Hardware: Server 1, Client 1, Client 2 (see section A.1).

6.2 TLS Persistent Results

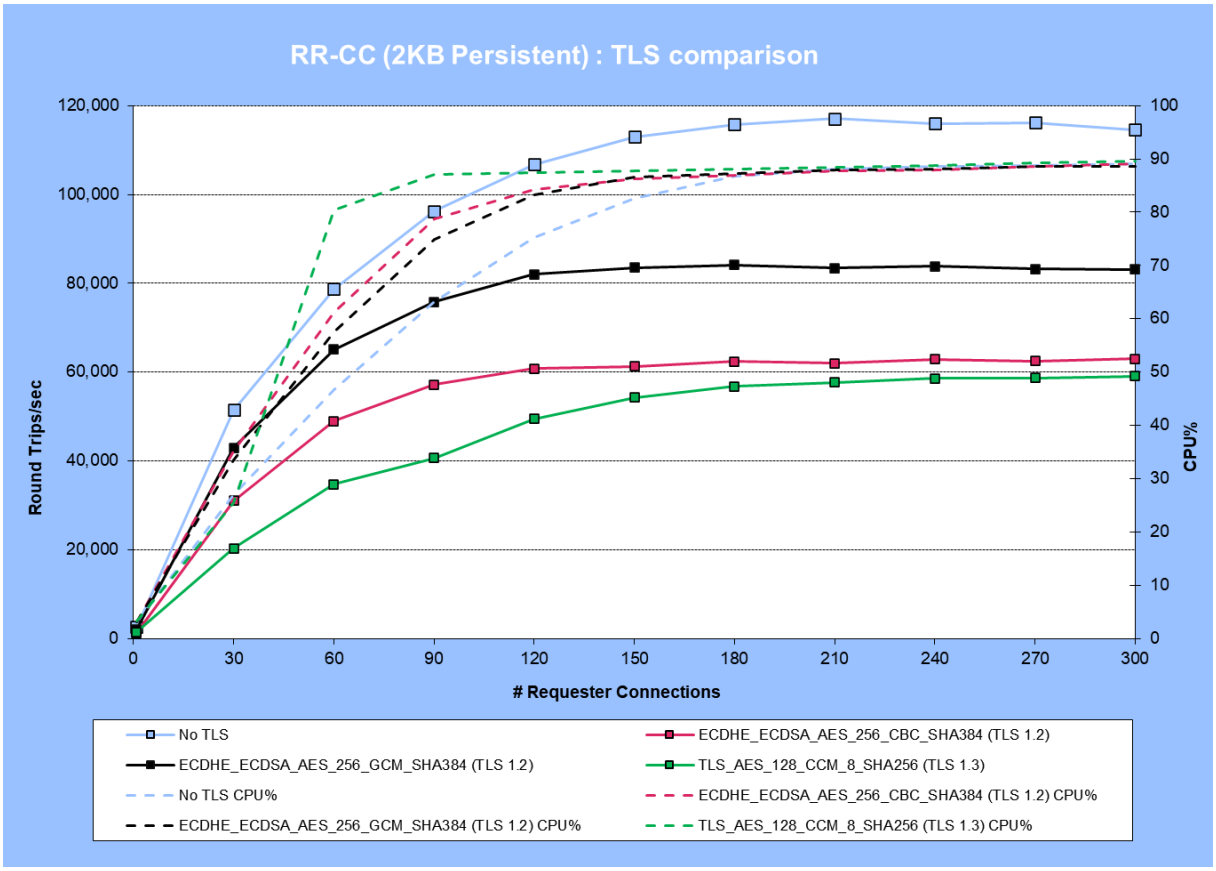

FIGURE 11 - PERFORMANCE RESULTS FOR RR-CC (2KB PERSISTENT) WITH TLS

Generally, the persistent TLS measurements showed a similar pattern to non-persistent tests (though the message rates were significantly lower throughout, as expected).

Table 11 and Table 12 show the peak throughputs for TLS 1.2 & TLS1.3 CipherSpecs.

TLS 1.2 CipherSpec	V9.4 GM Max Rate*	CPU%	Clients
No TLS	117,070	88	210
ECDHE_ECDSA_AES_256_CBC_SHA384	63,034	89	300
ECDHE_ECDSA_AES_256_GCM_SHA384	84,005	87	180
ECDHE_RSA_AES_256_CBC_SHA384	62,994	89	300
ECDHE_RSA_AES_256_GCM_SHA384	84,124	87	180

Round trips/sec

TABLE 11 - PEAK RATES FOR MQI CLIENT BINDINGS (2KB PERSISTENT) – TLS 1.2

TLS 1.3 CipherSpec	V9.4 GM Max Rate*	CPU%	Clients
No TLS	117,070	88	210
TLS_AES_128_CCM_8_SHA256	58,911	90	300
TLS_AES_256_GCM_SHA384	54,803	91	300
TLS_CHACHA20_POLY1305_SHA256	59,106	90	300
TLS_AES_128_GCM_SHA256	60,407	89	300
TLS_AES_128_CCM_SHA256	55,047	91	300

Round trips/sec

TABLE 12 - PEAK RATES FOR MQI CLIENT BINDINGS (2KB PERSISTENT) – TLS 1.3

6.2.1 Test setup.

Workload type: RR-CC (see section 3.1).

Hardware: Server 1, Client 1, Client 2 (see appendix A.1).

6.3 Effect of MQ Recovery Log Performance on TLS Comparisons

With persistent messaging, file I/O to the MQ recovery log is a significant throttling factor. This is evident in the TLS persistent messaging results as the ratio between non-TLS and TLS is lower. E.g. for 300 requester clients, non-persistent messaging, this is how a TLS 1.2 and a TLS 1.3 CipherSpec performs, in comparison to a non-TLS workload:

Scenario	**Rate**
No-TLS	371,209
ECDHE_ECDSA_AES_256_GCM_SHA384 (TLS 1.2)	226,220
TLS_AES_128_CCM_8_SHA256 (TLS 1.3)	136,839

Non-persistent ratio (No-TLS/TLS 1.2) = **1.64**
Non-persistent ratio (No-TLS/TLS 1.3) = **2.71**

For persistent messaging, the gap is closer:

Scenario	**Rate**
No-TLS	117,070
ECDHE_ECDSA_AES_256_GCM_SHA384 (TLS 1.2)	84,005
TLS_AES_128_CCM_8_SHA256 (TLS 1.3)	58,911

Persistent ratio (No-TLS/TLS 1.2) = **1.39**
Persistent ratio (No-TLS/TLS 1.3) = **1.99**

The persistent tests were run with the MQ recovery logs hosted on local, enterprise class NVMe devices, which are very fast. Hosting the recovery log off-box will result in lower

throughputs for persistent messaging and the comparison between non-TLS and TLS results will be more favourable (in throughput terms) though the CPU cost will be similar.

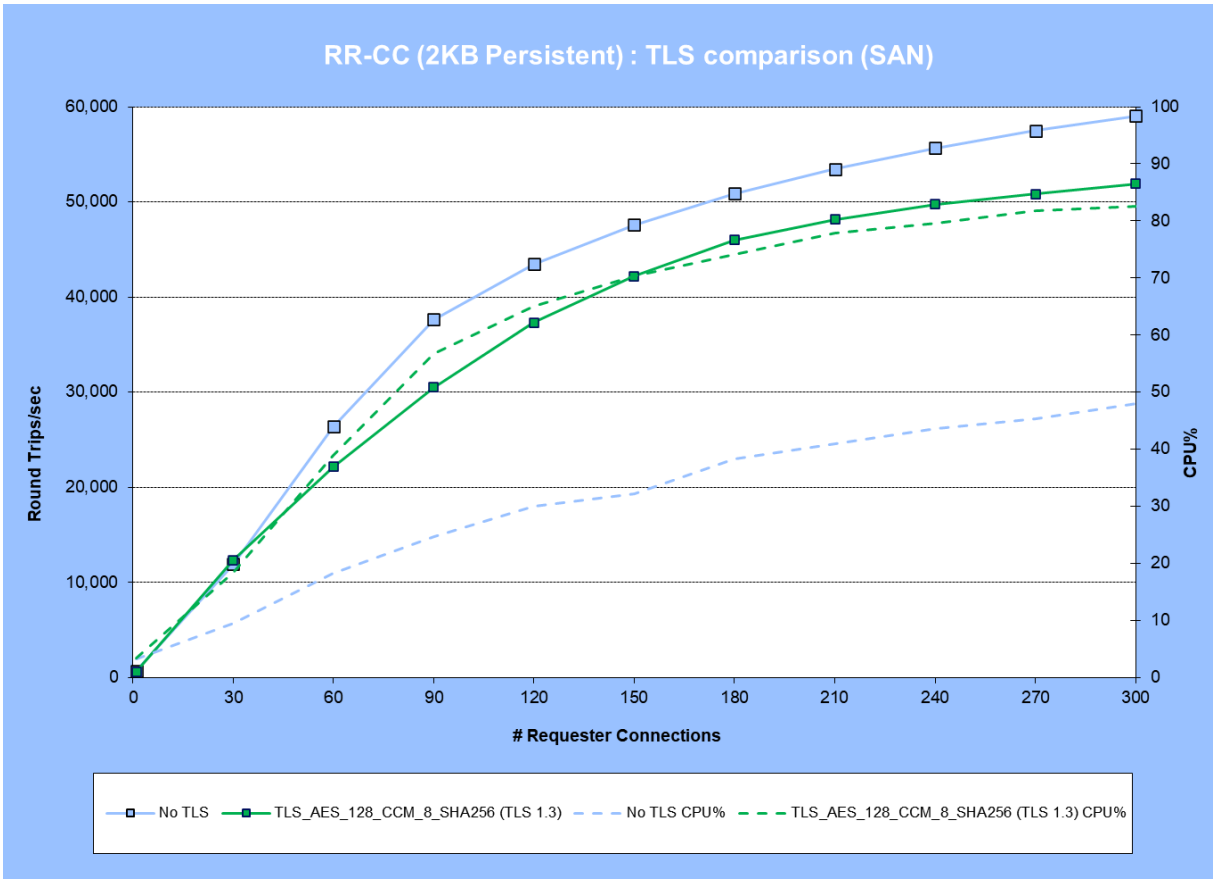

FIGURE 12 - PERFORMANCE RESULTS FOR RR-CC (2KB PERSISTENT) WITH TLS 1.3 (LOGGING TO SAN)

Figure 12 shows results for the RR-CC workload where the MQ recovery log is hosted on a SAN device (see appendix A.1 for details). In this case the throughputs are lower, as the recovery log file writes to the SAN filesystem are slower. As a result, the comparison (in throughput terms) between non-TLS and TLS is more favourable (see below) but note that the CPU overhead remains similar.

Scenario	**Rate**
No-TLS	59,043
TLS_AES_128_CCM_8_SHA256 (TLS 1.3)	51,897

Persistent ratio (No-TLS/TLS 1.3) = **1.14**

When evaluating TLS, you need to understand the performance capabilities of your infrastructure. Whilst there is a significant CPU cost incurred with encryption, if you have enough capacity the throughput impact may not be as much as the worst case for persistent message, as shown in Figure 11.

As for all performance evaluations, testing an environment as close as possible to that used in production is highly desirable. This will result in a much better understanding of the performance capabilities of the various components making up the environment your workload is running in.

6.3.1 Test setup.

Workload type: RR-CC (see section 3.1).

Hardware: Server 1, Client 1, Client 2 (see appendix A.1).

7 High Availability (HA) Test results

Highly availability for MQ typically minimises the time that a queue manager (and its messages) is not available to applications, and to remove single points of failure in the infrastructure where MQ runs. This typically requires:

- Fast detection of a failure of the running queue manager, and automatic recovery, often to a second physical system
- Persisting queue manager data to multiple physical disks

There are a number of options available to achieve high availability in MQ, including setting up general high availability managers, or deploying on MQ Appliances (see High availability configurations in the MQ doc).

This section will focus on the two technologies offered in software MQ; Multi Instance Queue Managers (MIQM) and replicated data queue managers (RDQM).

Both options provide automatic recovery, typically within seconds. However, the two have very different approaches to data resiliency.

With RDQM, data from the active queue manager instance is synchronously replicated to two other instances, so one of these instances can take over in the event of some failure. With MIQM the data is located on a remote NFS server, where another standby instance of the QM can access that same data in the event of a failover. MIQM does not by design replicate data, so if that data becomes generally inaccessible (e.g. a failure in the NFS server), neither queue manager will be able to continue. Typically, for higher resilience, the NFS server would be a clustered file system, backed by replicated storage.

The NFS server used in this report for MIQM does not provide replicated storage. This will impact the resiliency of the system and would not be equivalent in capability to a replicated RDQM solution. However, as the report shows, even non-replicated NFS storage does not perform as well as RDQM. If the MIQM data was to be synchronously replicated outside of MQ, to guarantee against data loss, the performance would be impacted further.

Although not included in this report, for MQ in Kubernetes, Native HA (NHA) is another option (and the alternative to using RDQM which is only applicable to non-container Linux deployments).

HA deployments most notably affect the performance of persistent messaging due to the recovery log writes occurring across the network (and replicated, in the case of RDQM), so only persistent messaging scenarios are shown in the following sections.

7.1 RR-CC Workload for HA with Unrestricted Replication Links

Figure 13 shows results for running a 2KB persistent messaging test in an HA setup using MIQM or RDQM, with a SAN test for comparison. Logging to SAN as a comparison is a fairer baseline as this represents a minimal at least a setup where the MQ storage for the queue manager is off box and could be backed by physical storage replication for redundancy.

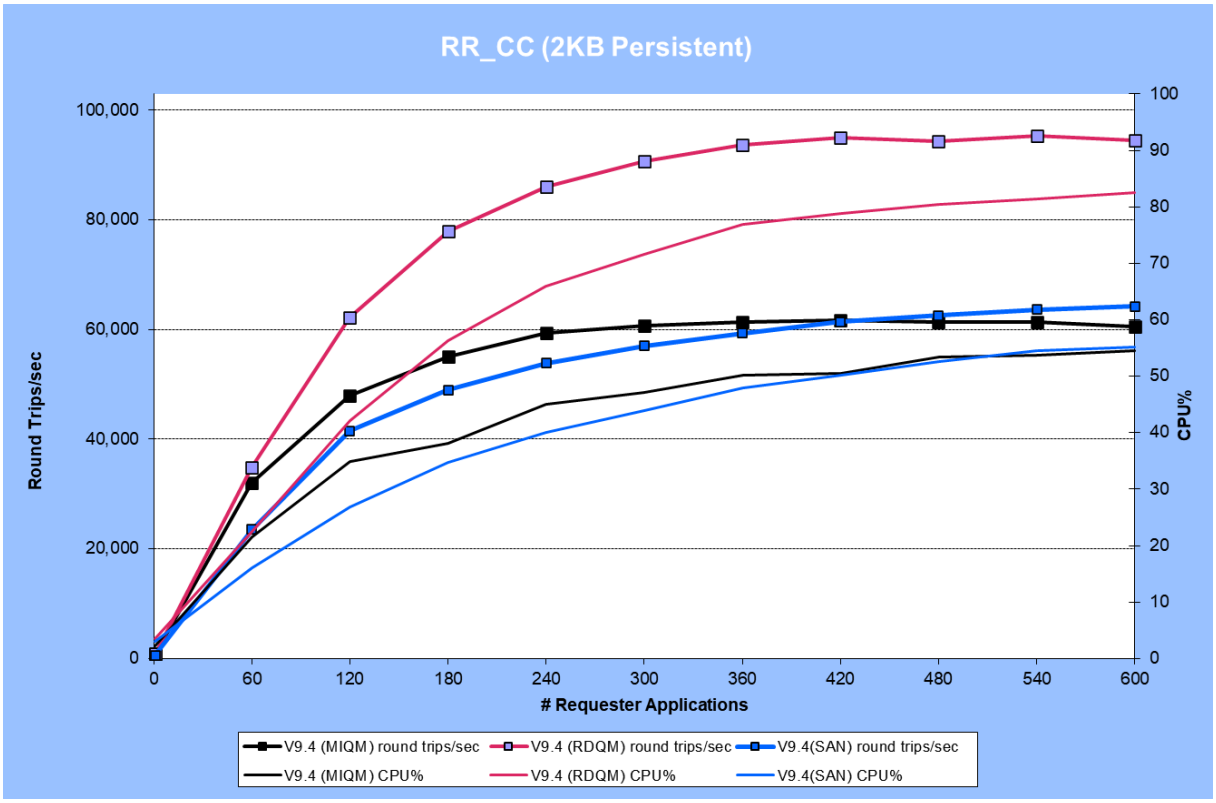

FIGURE 13 - 2KB PERSISTENT HA RESULTS

Care should be taken to review the infrastructure used in these tests. Results reflect the capabilities of the environment. HA persistent tests are highly sensitive to the capabilities of the network, and disk storage subsystems. These comparisons are to illustrate the differences that may be seen between different logging/HA approaches.

In the results above, RDQM out-performed the SAN tests, and more significantly also out-performed MIQM. Whilst a faster SAN environment might have resulted in better results (although maybe still without data replication), MIQM and RDQM are writing data over the network links (remember that MIQM is not replicating the data however, it is simply writing to a remote filesystem over NFS, such that the secondary QM can take-over using the same files, if necessary). Even though the RDQM scenario is providing a more robust HA solution it performed better in the tests than the MIQM solution.

Peak round trip rates for all message sizes tested can be seen in Table 13, with RDQM giving the best performance in all cases.

Test	V9.4 (MIQM)			V9.4 (RDQM)			V9.4 (SAN)		
	Max Rate*	CPU%	Clients	Max Rate*	CPU%	Clients	Max Rate*	CPU%	Clients
RR_CC (2KB Persistent)	61,669	50.51	420	95,403	81.47	540	64,249	55.14	600
RR_CC (20KB Persistent)	14,768	16.36	300	32,128	31.92	300	17,111	16.7	300
RR_CC (200KB Persistent)	1,945	7.75	120	3,836	13.53	120	2,658	8.86	120

Round trips/ sec

TABLE 13 - PEAK RATES FOR WORKLOAD RR-CC (2KB - MIQM/RDQM/SAN)

7.1.1 Test Setup

Workload type: RR-CC (see section 3.1).

Hardware:

SAN Test: Two client machines and a single server with MQ recovery logs hosted on SAN (see appendix section B.1)

MIQM Test: Two client machines, 2 QM hosts and an NFS Server (see appendix section 0)

RDQM Test: Two client machines and 3 QM hosts (see appendix section B.3)

7.2 Comparison of RDQM HA Results with Higher Replication Link Latencies.

This section will demonstrate the impact on performance of higher latency replication links. IBM supports RDQM HA replication links with latencies up to 5ms (see the doc here), though latencies of that magnitude may be unacceptable to your SLAs. If replication is required across higher latency links, then asynchronous replication via a DR solution is preferrable.

Data written across the replication links is done so in a synchronous fashion to ensure consistency of data in the event of a loss in communication with the primary queue manager requiring switchover to the secondary queue manager.

An HA solution will involve some physical separation of the hosts used to store replicated data. In RDQM, this may mean a significantly higher latency for the replication links. In an MIQM solution the higher latency may affect a solution replicating NFS hosted files.

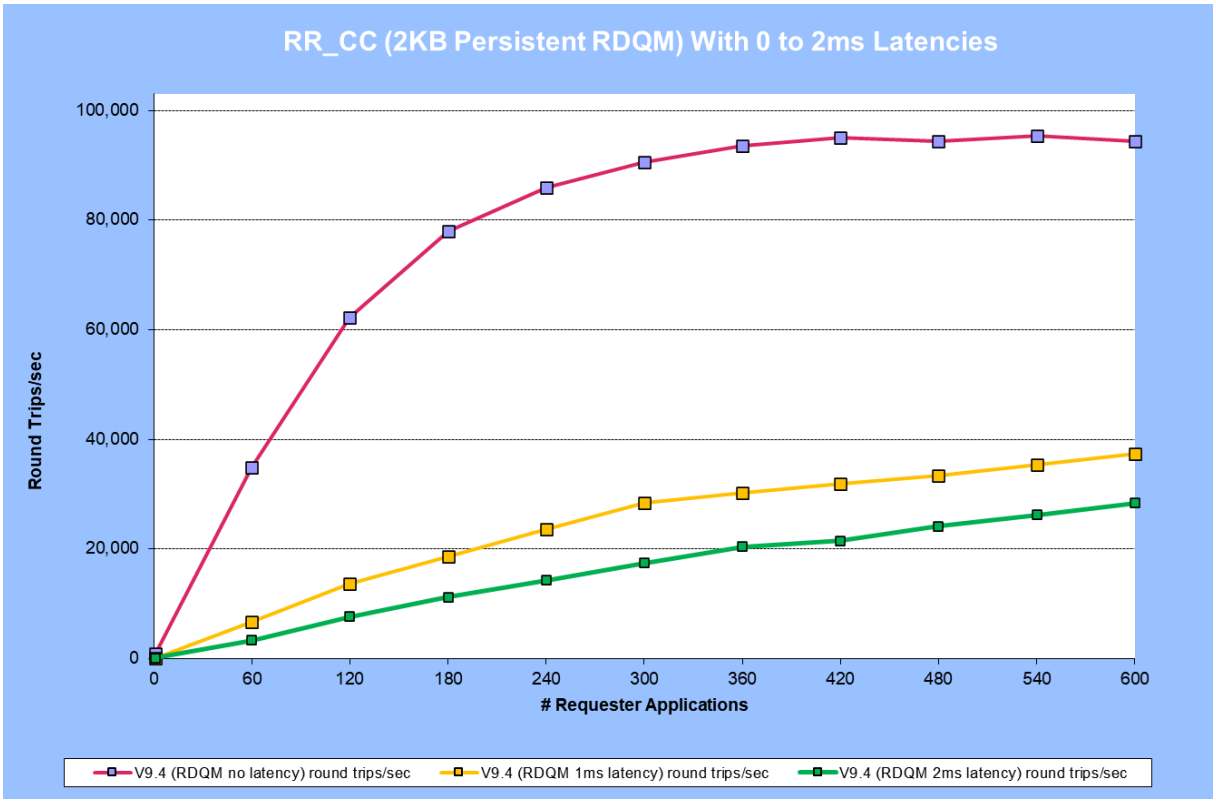

FIGURE 14 - 2KB PERSISTENT HA RESULTS FOR RDQM WITH 0 TO 2MS NETWORK LATENCY

Figure 14 shows the results of RDQM tests for 2KB messaging for where a delay was set on the replication links only (the separate links to machines hosting the client applications are unhindered).

The delays set were:

- No Latency : Replication links are unrestricted.
- 1ms Latency : A 500µs delay was set on the inbound and outbound path of the link, resulting in a 1ms round-trip delay
- 2ms Latency : A 1ms delay was set on the inbound and outbound path of the link, resulting in a 2ms round-trip delay

Higher latency links dramatically reduces the round-trips/sec, but these tests still scale in an orderly fashion as more requester applications are started. Writes to the MQ recovery log are typically the limiting factor in persistent messaging and certainly so in the cases with the higher latency links.

The MQ logger will aggregate log writes more as the network latency increases however, minimising the impact of the delay by utilising more of the bandwidth of the link with larger write. You can see the this by monitoring log write sizes using the amqsrua sample program, e.g. on a queue manager called QM1:

```
amqsrua -m QM1 -c DISK -t Log
```

In our unrestricted case, the latency of the link is very low, as the network switch resides in the same rack as the hosts it is connecting.

All deployments will be different, so it's impossible to predict the affect that a higher latency will have. If the network latency of the links between the applications and the queue managers is high for instance, then a higher latency on the HA links may not have such a big effect. As with all performance considerations it's imperative that you carry out your own tests to assess the infrastructure you intend to deploy on.

Peak round trip rates for all message sizes tested with 1ms and 2ms replication link latencies can be seen in Table 14 & Table 15.

Test	V9.4 (RDQM) Max Rate*	CPU%	Clients
RR_CC (2KB Persistent) - 1ms Network latency	37,330	35.64	600
RR_CC (20KB Persistent) - 1ms Network latency	9,088	13.26	300
RR_CC (200KB Persistent) - 1ms Network latency	1,152	7.26	120

Round trips/ sec

TABLE 14 - PEAK RATES FOR WORKLOAD RR-CC (2KB - RDQM WITH 1MS NETWORK LATENCY)

Test	V9.4 (RDQM) Max Rate*	CPU%	Clients
RR_CC (2KB Persistent) - 2ms Network latency	28,390	27.31	600
RR_CC (20KB Persistent) - 2ms Network latency	7,551	11.75	300
RR_CC (200KB Persistent) - 2ms Network latency	1,016	6.53	135

Round trips/ sec

TABLE 15 - PEAK RATES FOR WORKLOAD RR-CC (2KB - RDQM WITH 2MS NETWORK LATENCY)

7.2.1 Test Setup

Workload type: RR-CC (see section 3.1).

Hardware:

Two client machines and 3 QM hosts (see appendix section B.3)

7.3 Circular Logging vs Linear Logging.

All the tests in section 7 used linear logging instead of circular logging for consistency of HA test scenarios in the lab (where NHA is also tested, and which requires the use of linear logging). There is little difference in the performance of linear and circular logging (see Figure 15).

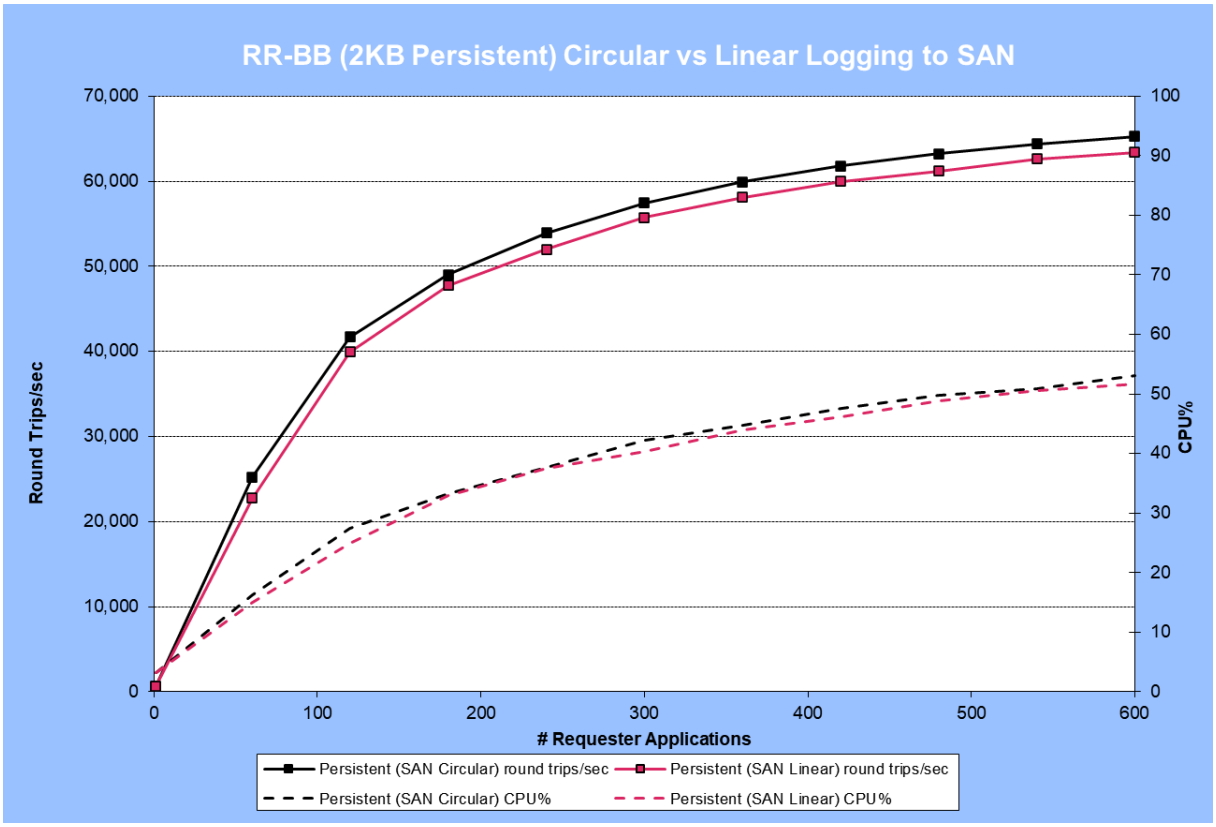

FIGURE 15 - CIRCULAR VS LINEAR LOGGING TO SAN

7.3.1 Test Setup

Workload type: RR-CC (see section 3.1).

Hardware:

Two client machines and a single server with MQ recovery logs hosted on SAN (see appendix section B.1)

Appendix A: Non-HA Test Configurations

A.1 Hardware/Software – Set1

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, client1 & client2 are three identical machines:

- ThinkSystem SR630 V2– [7Z71CTO1WW]
- 2 x 16 core CPUs.
  Core: Intel(R) Xeon(R) Gold 6346 CPU @ 3.10GHz
- 256GB RAM
- Queue manager recovery log and queue data stored locally on 2 x 3.2TB NVMe SSDs (KCM61VUL3T20) in RAID 0 array, unless otherwise specified.
- 100Gb ethernet adapters connect all three machines via an isolated performance LAN (except for message compression tests where 10Gb links were used).
- Hyper-Threading is enabled but Turbo Boost is disabled. This is to assist with achieving the best performance that is also consistent.

SAN Infrastructure:

- IBM 2498-F48 fibre channel SAN switch (16Gb/s ports)
- IBM SAN Volume Controller (2145-SV1) with 256GB RAM
- IBM Flash System 900 Storage.

A.1.2 Software

- Red Hat Enterprise Linux Server release 8.9 (Ootpa)
- JMSPerfHarness test driver (see Appendix D:)
- MQ-CPH MQI test driver (see Appendix D:)
- IBM MQ V9.4

A.1.3 Software

Red Hat Enterprise Linux Server release 8.9 (Ootpa)

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

IBM MQ V9.4

A.2 Tuning Parameters Set for Measurements in This Report

The tuning detailed below was set specifically for the tests being run for this performance report but in general follow the best practises.

A.2.1 Operating System

A good starting point is to run the IBM supplied program mqconfig. The following Linux parameters were set for measurements in this report.

/etc/sysctl.conf

```
fs.file-max = 19557658
net.ipv4.ip_local_port_range = 1024 65535
net.core.rmem_max = 2147483647
net.core.wmem_max = 2147483647
net.ipv4.tcp_rmem = 4096 87380 2147483647
net.ipv4.tcp_wmem = 4096 65536 2147483647
vm.max_map_count = 1966080
kernel.pid_max = 655360
kernel.msgmnb = 131072
kernel.msgmax = 131072
kernel.msgmni = 32768
kernel.shmmni = 8192
kernel.shmall = 18446744073692774399
kernel.shmmax = 18446744073692774399
kernel.sched_latency_ns = 2000000
kernel.sched_min_granularity_ns = 1000000
kernel.sched_wakeup_granularity_ns = 400000
```

/etc/security/limits.d/mqm.conf

```
@mqm soft nofile 1048576
@mqm hard nofile 1048576

@mqm soft nproc  1048576
@mqm hard nproc  1048576
```

```
NFS mount for the MQ recovery log in NFS tests used the following
parameters:
rsize=1048576,wsize=1048576
```

A.2.2 IBM MQ

The following parameters are added or modified in the qm.ini files for the tests run in section 4 of 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 (200K & 2MB), the queue buffers were increased further to:

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

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.

All client channels were configured with SHARECNV(1), which is the recommendation for performance.

Appendix B: HA Test Configurations

B.1 SAN Storage Baseline Topology

The SAN tests used to compare with MIQM and RDQ tests used the following topology.

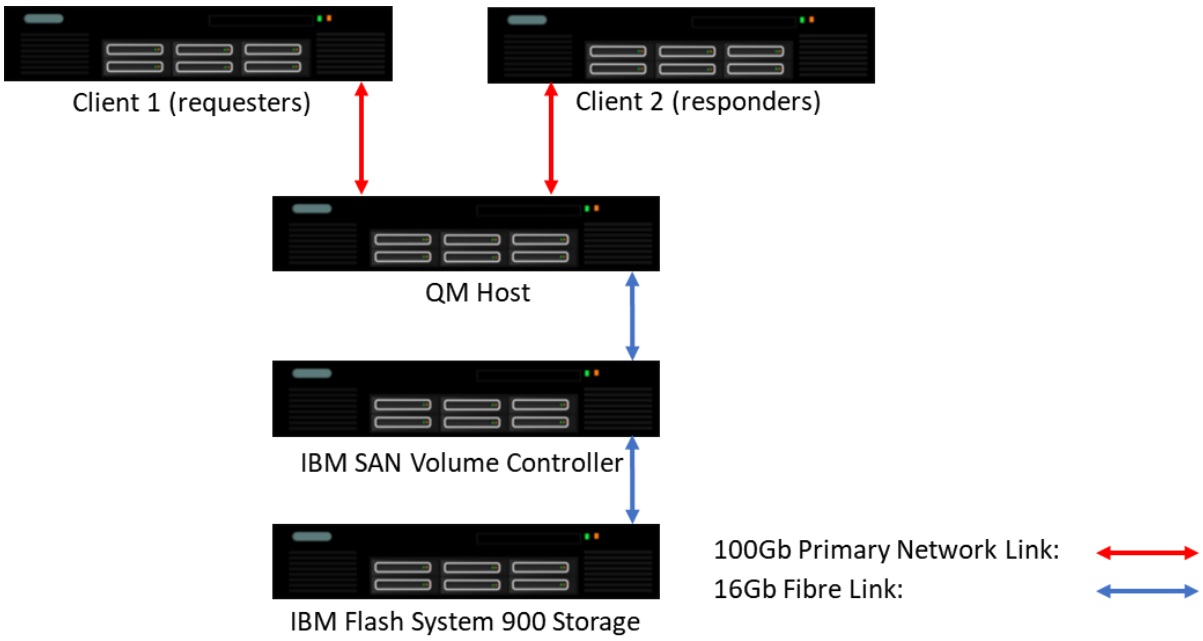

FIGURE 16 - SAN TEST TOPOLOGY

B.1.1 Hardware

QM Host:

- ThinkSystem SR630 V2– [7Z71CTO1WW]
- 2 x 16 core CPUs.
  Core: Intel(R) Xeon(R) Gold 6346 CPU @ 3.10GHz
- 256GB RAM
- Queue manager recovery log and queue data on SAN.
- 100Gb ethernet adapter on an isolated performance LAN.
- Hyper-Threading is enabled but Turbo Boost is disabled. This is to assist with achieving the best performance that is also consistent.

Client1 & Client2:

- ThinkSystem SR630 - [7X02CTO1WW]
- 2 x 12 core CPUs.
  Core: Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz
- 192GB RAM

SAN Infrastructure:

- IBM 2498-F48 fibre channel SAN switch (16Gb/s ports)
- IBM SAN Volume Controller (2145-SV1) with 256GB RAM
- IBM Flash System 900 Storage.

B.1.2 Software

- Red Hat Enterprise Linux Server release 8.5 (Ootpa)
- MQ-CPH MQI test driver (see Appendix D:)
- IBM MQ V9.4

B.2 MIQM Topology

For the MIQM tests the file system exported by the NFS server to host the MQ logs and queues, was deployed on 2 x 3.2TB NVMe SSDs (KCM61VUL3T20) in a RAID 0 array. Links between the applications and the MIQM QM hosts were 100Gb on the primary subnet, whilst the NFS links were 100Gb on a separate subnet.

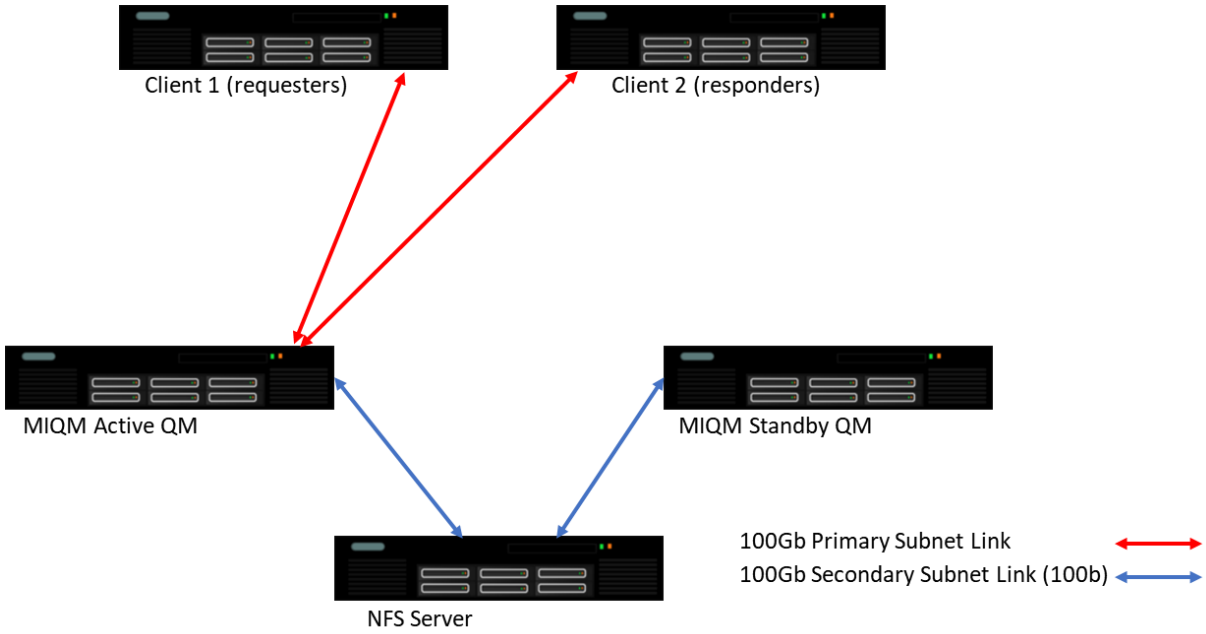

FIGURE 17 - MIQM TEST TOPOLOGY

B.2.1 Hardware

Active/Standby QM hosts and NFS Server:

- ThinkSystem SR630 V2– [7Z71CTO1WW]
- 2 x 16 core CPUs.
  Core: Intel(R) Xeon(R) Gold 6346 CPU @ 3.10GHz
- 256GB RAM
- Queue manager recovery log and queue data on SAN.
- 100Gb ethernet adapter on an isolated performance LAN.
- Hyper-Threading is enabled but Turbo Boost is disabled. This is to assist with achieving the best performance that is also consistent.

Client1 & Client2:

- ThinkSystem SR630 - [7X02CTO1WW]
- 2 x 12 core CPUs.
  Core: Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz
- 192GB RAM

B.2.2 Software

- Red Hat Enterprise Linux Server release 8.5 (Ootpa)
- MQ-CPH MQI test driver (see Appendix D:)
- IBM MQ V9.4

B.3 RDQM Topology

For RDQM testing, all three RDQM nodes were of type 1 (see machine types, below), and the two application hosts were of type 2. The DRBD volume groups were deployed on 2 x 3.2TB NVMe SSDs (KCM61VUL3T20) in a RAID 0 array. Links between the applications and the RDQM nodes were 100Gb on the primary subnet, whilst the RDQM data replications links were 100Gb on a separate secondary subnet. Each RDQM node had a single Pacemaker address (HA_Primary), utilising the 100Gb link.

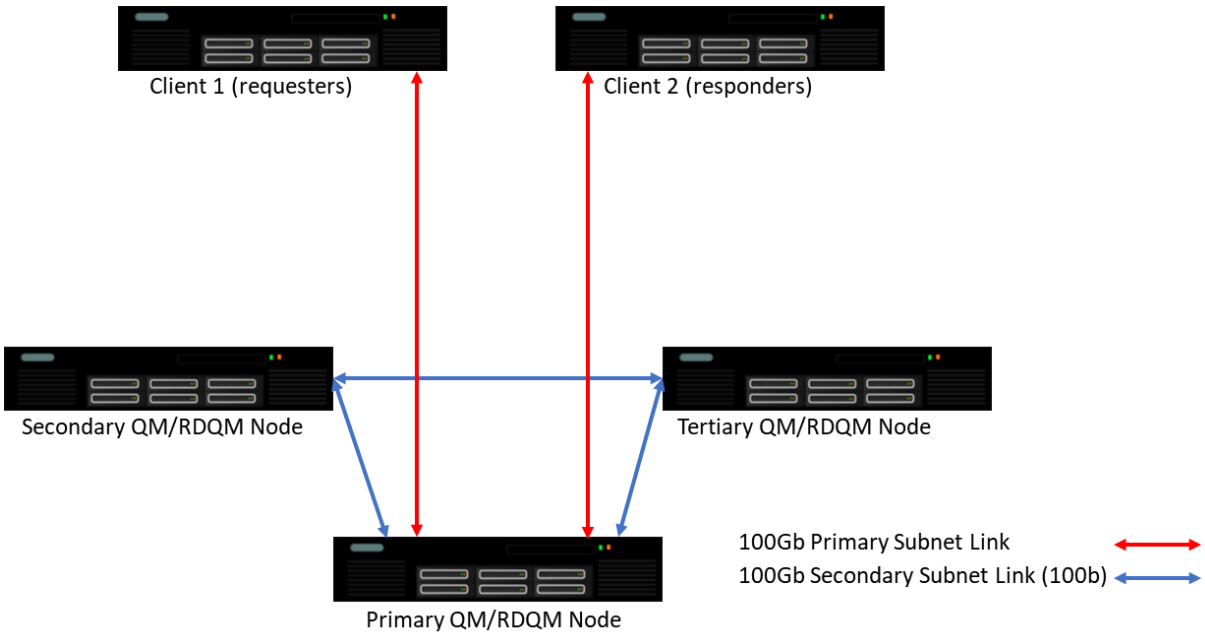

FIGURE 18 - RDQM TEST TOPOLOGY

B.3.1 Hardware

Primary/Secondary/Tertiary QM hosts:

- ThinkSystem SR630 V2– [7Z71CTO1WW]
- 2 x 16 core CPUs.
  Core: Intel(R) Xeon(R) Gold 6346 CPU @ 3.10GHz
- 256GB RAM
- Queue manager recovery log and queue data on SAN.
- 100Gb ethernet adapter on an isolated performance LAN.
- Hyper-Threading is enabled but Turbo Boost is disabled. This is to assist with achieving the best performance that is also consistent.

Client1 & Client2:

- ThinkSystem SR630 - [7X02CTO1WW]
- 2 x 12 core CPUs.
  Core: Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz

- 192GB RAM

B.3.2 Software

- Red Hat Enterprise Linux Server release 8.5 (Ootpa)
- MQ-CPH MQI test driver (see Appendix D:)
- IBM MQ V9.4

Appendix C: 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 D: Resources

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

IBM MQ Performance: Best Practises, and Tuning Paper:
https://ibm-messaging.github.io/mqperf/MQ_Performance_Best_Practices_v1.0.1.pdf

IBM MQ Test Harnesses Launch Page (includes links to containerised versions of MQ-CPH & JMSPerfHarness) :
Test Harnesses.

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

Tutorial: MQ-CPH_Introduction.pdf

JMSPerfHarness (The IBM MQ Performance Harness for JMS)
https://github.com/ot4i/perf-harness

Tutorial: jmsperfharness_tutorial1.md