IBM MQ V9.2 for Linux (x86-64 platform) Performance Report

Version 1.0 - February 2021

Paul Harris
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, February 2021.

This edition applies to *IBM MQ V9.2* (and to all subsequent releases and modifications until otherwise indicated in new editions).

Note to U.S. Government Users

Documentation related to restricted rights.

Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

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 V9.2. 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 V9.2.

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 v9.2 for Linux on x86_64.
- Want to understand the performance limits of IBM MQ v9.2 for Linux on x86_64.
- Want to understand what actions may be taken to tune IBM MQ v9.2 for Linux on x86_64.

The reader should have a general awareness of the Linux 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. 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 V9.2 is a long term service (LTS) release of MQ, which includes features made available in the V9.1.1, V9.1.2, V9.1.3, V9.1.4 & V9.1.5 continuous delivery (CD) releases. Those CD releases are mainly functional in nature, but there are some significant improvements to performance as well

- Queue manager restart times (see section 1)
- Improved switch/fail-over times (see section 0)

The release highlights section of this report also includes detail on Uniform Cluster function, , first introduced in V9.1.2 CD and rolled into 9.2 LTS.

Performance data presented in this report does not include release to release comparisons, but all tests run showed equal or better performance than V9.0 & V9.1 releases 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.

1 Release Highlights

1.1 Queue manager restart time improvements

MQ V9.2 includes significant improvements to queue manager restart times, introduced in the V9.1.1 CD release of MQ.

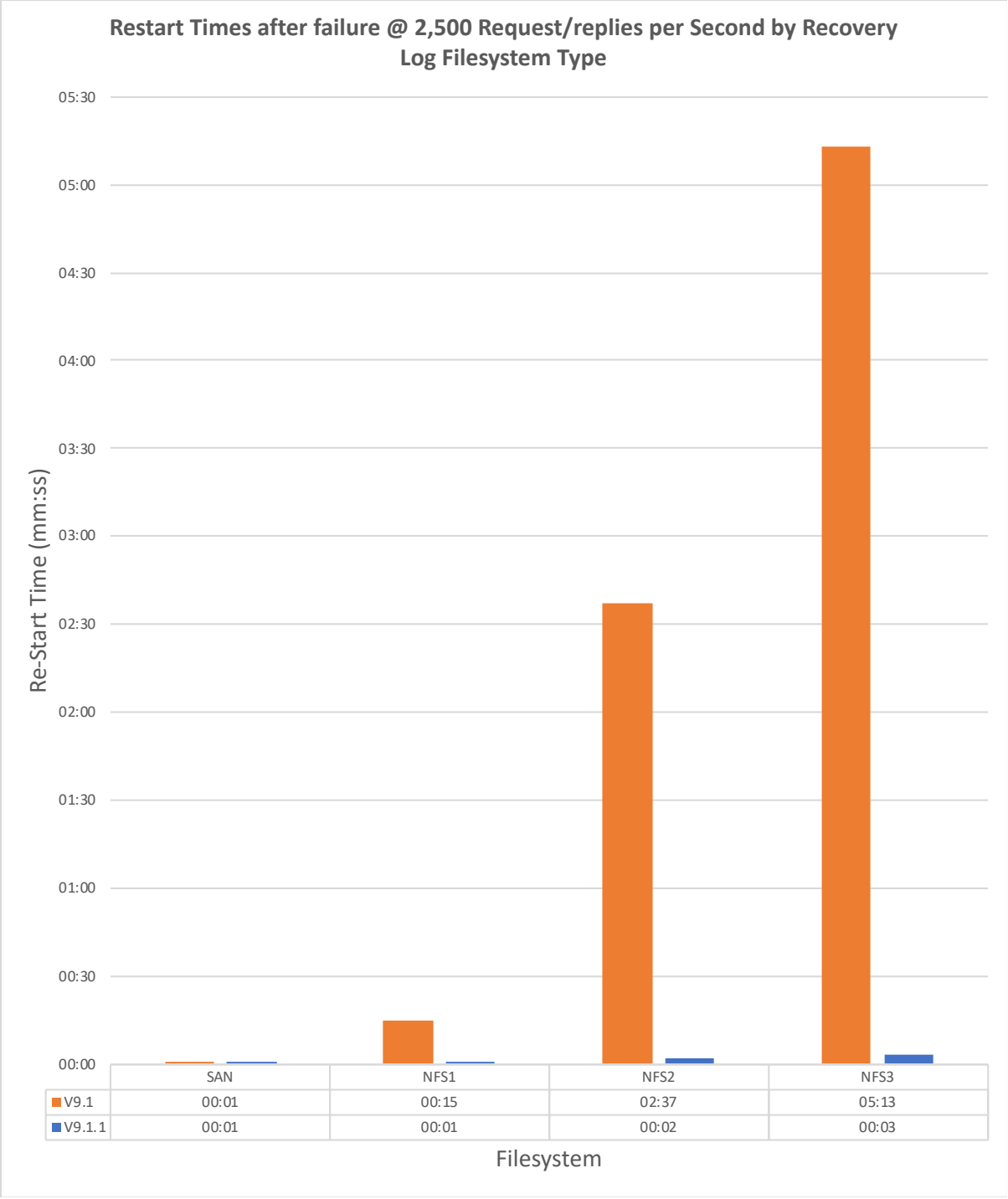


FIGURE 1 - RESTART TIMES FOR BUSY QM

SAN: SAN hosted file system, using an IBM San Volume Controller (SVC), via 8Gb fibre links.

NFS1: NFS (V4) hosted on RAID cached disks via a dedicated* 10Gb network link.

NFS2: NFS (V4) hosted on RAID cached disks via a dedicated* 10Gb network link with an additional 500us delay on network added in each direction.

NFS3: NFS (V4) hosted on RAID cached disks via a dedicated* 10Gb network link with an additional 1ms delay on network added in each direction.

Figure 1, shows the restart times for a busy queue manager across a number of different file systems (see above) hosting the queue files and recovery log. Tests measure restart time when there has been a failure of some type, causing the queue manager to abruptly end, with potentially in-flight transactions (in our test scenarios we killed the queue manager processes).

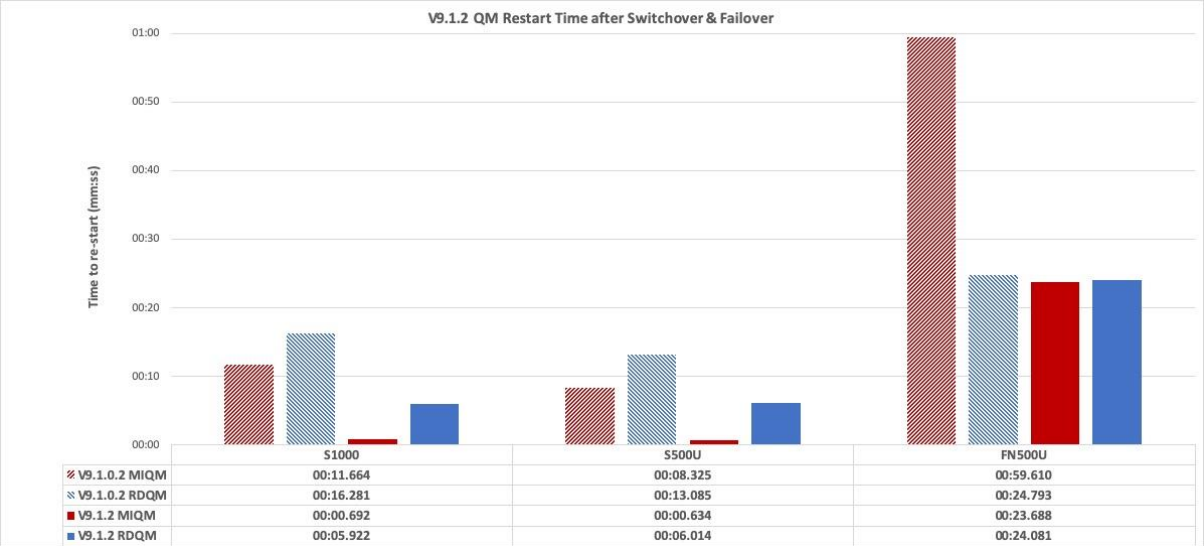
Re-start times are improved significantly for slower file systems such as NFS, but changes in V9.1.1 also improve re-start times where there are deep queues, regardless of where the recovery log is located.

A paper with fuller details has been published on the MQ performance github site: https://ibm-messaging.github.io/mqperf/Queue%20Manager%20Restart%20Times.pdf

1.1 Improved Switch/Fail-over times

Improvements introduced in the V9.1.2 CD release of MQ focused on the processing of client disconnections, in particular the time it takes to disconnect significant numbers of clients when switching queue managers or processing the fail-over of a queue manager.

Figure 2 shows a comparison of V9.1.0.2 vs V9.1.2 re-start times in three different scenarios:



	S1000	S500U	FN500U
V9.1.0.2 MIQM	00:11.664	00:08.325	00:59.610
V9.1.0.2 RDQM	00:16.281	00:13.085	00:24.793
V9.1.2 MIQM	00:00.692	00:00.634	00:23.688
V9.1.2 RDQM	00:05.922	00:06.014	00:24.081

FIGURE 2 – TIME TO RESTART AFTER SWITCHOVER OR FAILOVER (MIQM & RDQM)

	Test Description
S1000	Switch-over with 1000 applications running at a set rate of 1 Put/Get per sec (total 1000 round trips/sec)
S500U	Switch-over with 500 applications running unrated* (MIQM total rate ~50,000 Put/Gets per sec. RDQM total rate ~85,000 Put/Gets per sec).
FN500U	Fail-over (network) with 500 applications running unrated (MIQM total rate ~50,000 Put/Gets per sec. RDQM totalrate ~85,000 Put/Gets per sec). nfsv4leasetime & nfsv4gracetime reduced to 10secs

*unrated applications execute Put/Get requests at the maximum number/per second rather than a set rate (e.g. 1 per second).

Further details can be found in this blog article:
https://community.ibm.com/community/user/imwuc/viewdocument/improved-switchfail-over-times-in

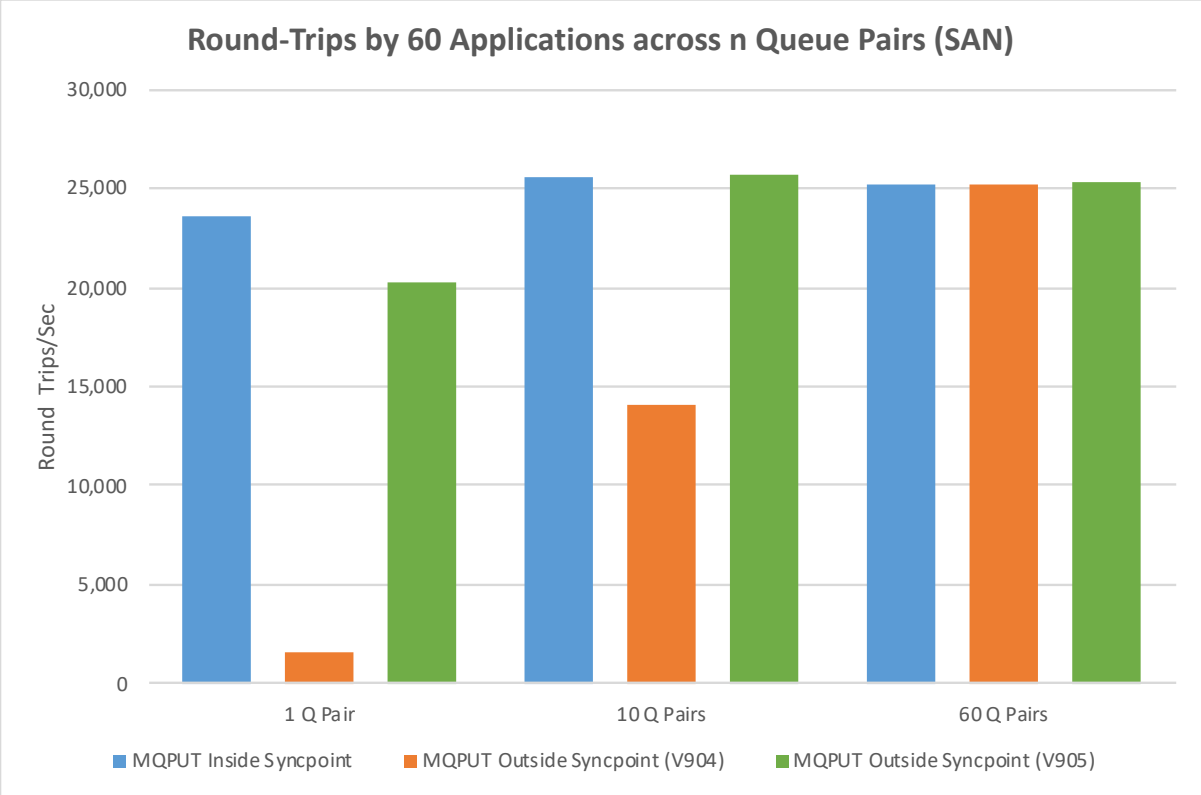


FIGURE 3 – EFFECT OF SYNCPOINT BY NUMBER OF QUEUES

Figure 3 shows the effect of reducing queue locking by spreading the load across a number of queue pairs (REQUEST Q/REPLY Q). All tests use 60 requester applications. When the workload is driven through a single pair of queues, the non-syncpoint case has a low throughput (not much better than the test using 1 requester in chart 1), as each MQPUT queues up behind the previous one to that queue, with a forced log write being executed within the scope of the queue lock. Using syncpoints alleviates this issue, allowing for more concurrency. As we increase the number of queue pairs, the locking becomes less of an issue, until, at 60 pairs of queues, where there are only 2 requester applications per queue pair, the non-syncpoint case is not much less than using syncpoints. Once again, the V.9.0.5 test case, with PUTs outside of syncpoints matches the explicit syncpoint scenario.

1.2 Uniform Cluster

MQ Uniform cluster, first introduced in V9.1.2 CD and rolled into 9.2 LTS enables capabilities to horizontally scale applications across a small set of similar horizontally scaled queue managers. Applications can be moved as necessary by the cluster to balance the workload as additional cluster members are started.

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.2.0/com.ibm.mq.con.doc/q132725_.html

In this example a simple point-to point messaging test demonstrates how we can start additional cluster members to distribute the load as the rate of message delivery is increased, with the existing applications being balanced across all available cluster members automatically.

1.2.1 Test Topology:

In the test that follows, 12 queue managers ('QMs') in the uniform cluster 'UC01' were used. Each QM was started, when required.

Each queue manager is restricted to 4 cores. With 6 QMs on each of two 24 core machines (QMHost1 & QMHost2 below), in its own cgroup.

This simulates the addition of extra compute capacity added as required to horizontally scale the messaging layer, as you would expect in many virtual deployments such as cloud VMs and containers (Linux cgroups is the same technology used to control the resources used by containers in solutions such as RedHat OpenShift).

For this simple test all application threads are created using the JMS component of the PerfHarness test tool from the start, although in real life these would often be expected to ramp up as the load on the system increases. As we're using MQ's Uniform Cluster topology the application threads will automatically be rebalanced across all available queue managers in the cluster. This automatically maximises the capacity of the horizontally scaled set of queue managers.

The PerfHarness sender process was started on host AppHost1 and the PerfHarness receiver process was start on host AppHost2.

All machines in the test are connected via 40Gb network links.

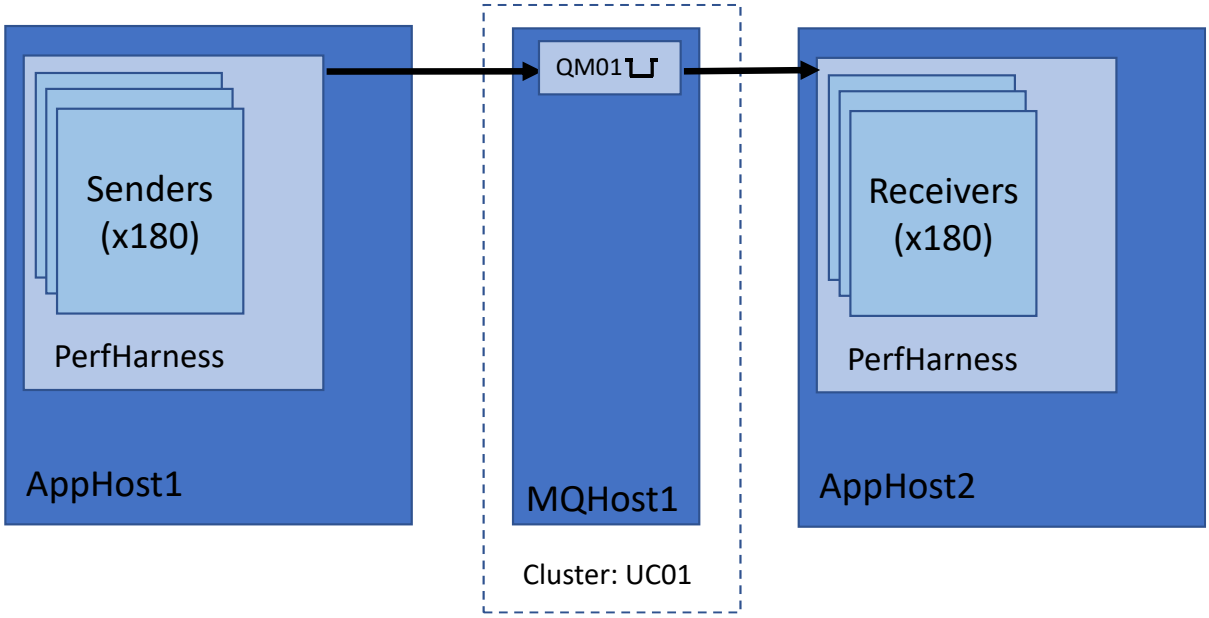


FIGURE 4 : TEST TOPOLOGY - START OF TEST

Again, for the simplicity of the test, all 12 of the uniform cluster queue managers are created in advance, although they are started individually over the duration of the test as the load ramps up. Pre-creating the queue managers is not necessary in a real deployment as MQ supports creating and joining queue managers into a cluster dynamically, and for applications to dynamically load new connection information. Figure 4 shows the topology of the test at the start, with one queue manager started. Figure 5 Figure 5 shows the topology of the test at the end when all 12 queue managers have been started to absorb the increase messaging rate.

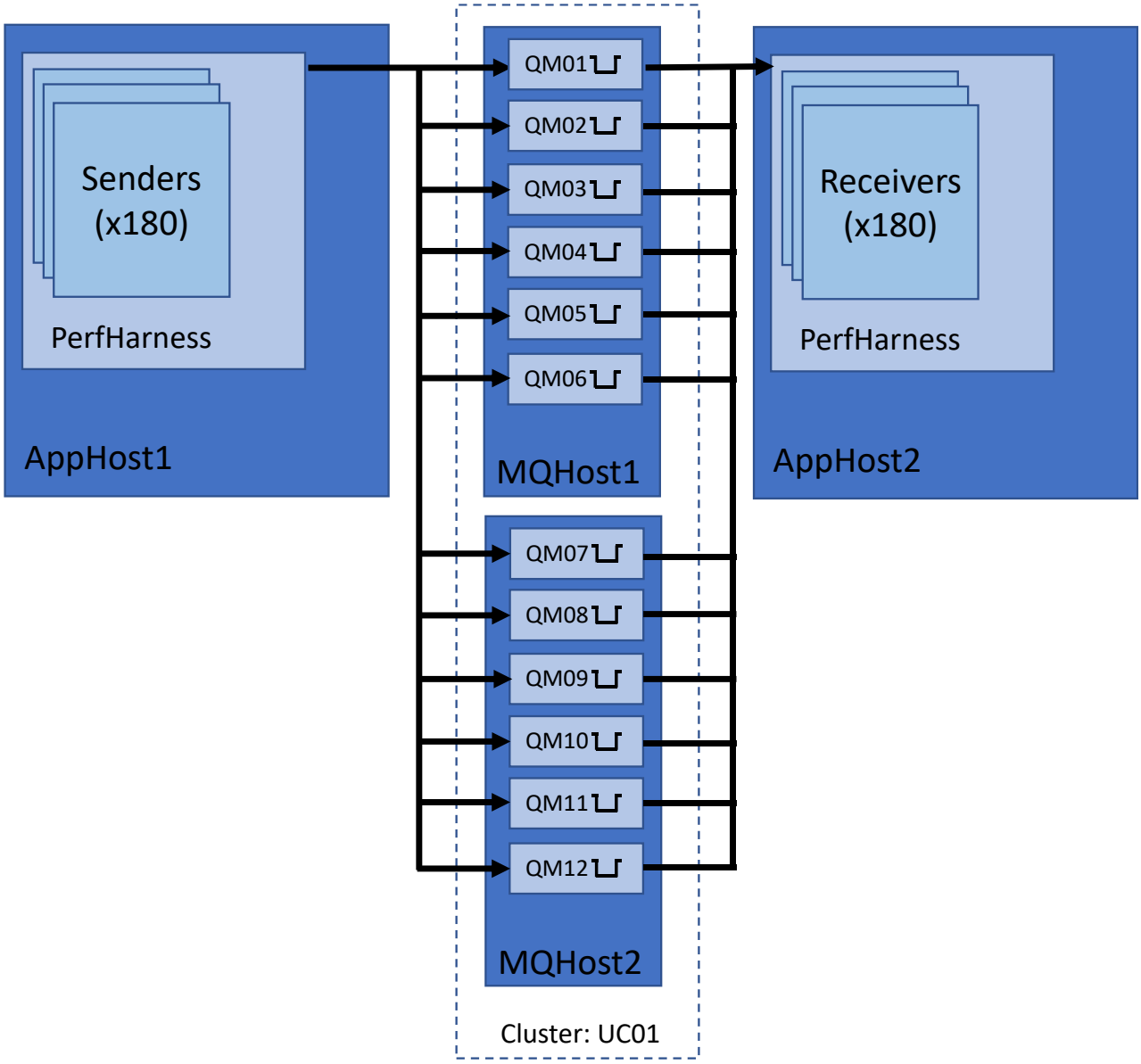


FIGURE 5: TEST TOPOLOGY - END OF TEST

1.2.2 Test Scenario

The objective of the test is to initially establish all the applications with a single queue manager active and progressively increase the messaging load to the point that the queue manager becomes the limiting factor. We periodically start up additional queue managers and see how the messaging traffic is horizontally balanced to increase capacity. The steps are:

1. A single QM (UC01QM01) on QMHost1 is started in its own cgroup slice, restricting it to 4 cores of CPU.
2. 180 re-connectable sender application threads are started on AppHost1, putting 2KB non-persistent messages at a rate of 1 message per second (per thread) to a single queue. Each thread connects to MQ setting its APPLTAG to 'mqperf.sender'
3. 180 re-connectable receiver application threads are started on AppHost2. Each thread connects to MQ setting its APPLTAG to 'mqperf.receiver'

4. The rate of the *existing* senders is increased by 115 messages / second (per sender)
5. An additional uniform cluster member is started
6. Repeat from 4 until all 12 QMs are started.

The PerfHarness applications use a json CCDT to connect to the queue managers using a QM group of 'UC01QM'. All the cluster QMs are in the UC01QM group and as only UC01QM01 is up when the receivers and senders are started, they all connect to that QM. Again, this is for simplicity. In real life you can add new entries into a CCDT as you create new queue managers in the uniform cluster dynamically. The connected applications will automatically reload the new information as they need to.

The *only* controls on the test are to increase the message rate of the *existing* senders and to start additional uniform cluster QMs. The increase of rate before each additional QM was started caused the existing QMs to be restricted by the cgroup CPU resource limit. When an additional QM is started, the senders and receivers are automatically re-balanced, so the rate achieved increases.

The rate achieved by the senders was measured after each increase and after each additional QM had been started.

1.2.3 Results

Message rates and CPU% consumed by the two MQ host machines are shown in the table below. Each time the target rate is increased (i.e. the total attempted message delivery rate of all senders), the QMs that are started reach the limit of their cgroup.slice allocation. The target rate is then achieved when the next QM is started.

Target Rate	Rate Achieved	#QMs	MQHost1 CPU%	MQHost2 CPU%
20,700	20,700	1	10.96	0.00
41,400	21,070	1	16.06	0.06
41,400	41,402	2	26.93	0.02
62,100	42,244	2	32.63	0.00
62,100	62,101	3	44.59	0.00
82,800	63,108	3	49.13	0.00
82,800	82,809	4	63.02	0.00
103,500	84,896	4	65.63	0.00
103,500	103,495	5	79.76	0.00
124,200	106,077	5	82.00	0.00
124,200	124,192	6	94.53	0.00
144,900	125,901	6	95.75	0.00
144,900	144,892	7	94.45	9.12
165,600	146,211	7	95.06	9.13
165,600	165,595	8	94.28	23.02
186,300	165,589	8	94.25	23.06
186,300	186,293	9	93.33	42.60
207,000	186,347	9	93.81	42.69
207,000	206,974	10	96.45	56.05
227,700	206,974	10	95.25	57.81
227,700	227,670	11	96.16	73.03
248,400	227,675	11	96.00	72.81
248,400	248,338	12	96.88	86.95

Table 1 : Uniform Cluster Result

Figure 6 below shows a plot of the message rate and CPU consumption of the QM hosts. Two points on the graph are indicated to show points where the message rate was increased and where a QM was subsequently started to accommodate the additional demand. As the test progresses the fixed increment in rate (representing around 85% of 1 QMs messaging capability) can be accommodated more by the larger number of QMs already started, so a noticeable step in increase can be seen at that point, followed by a further increase when the additional QM is started.

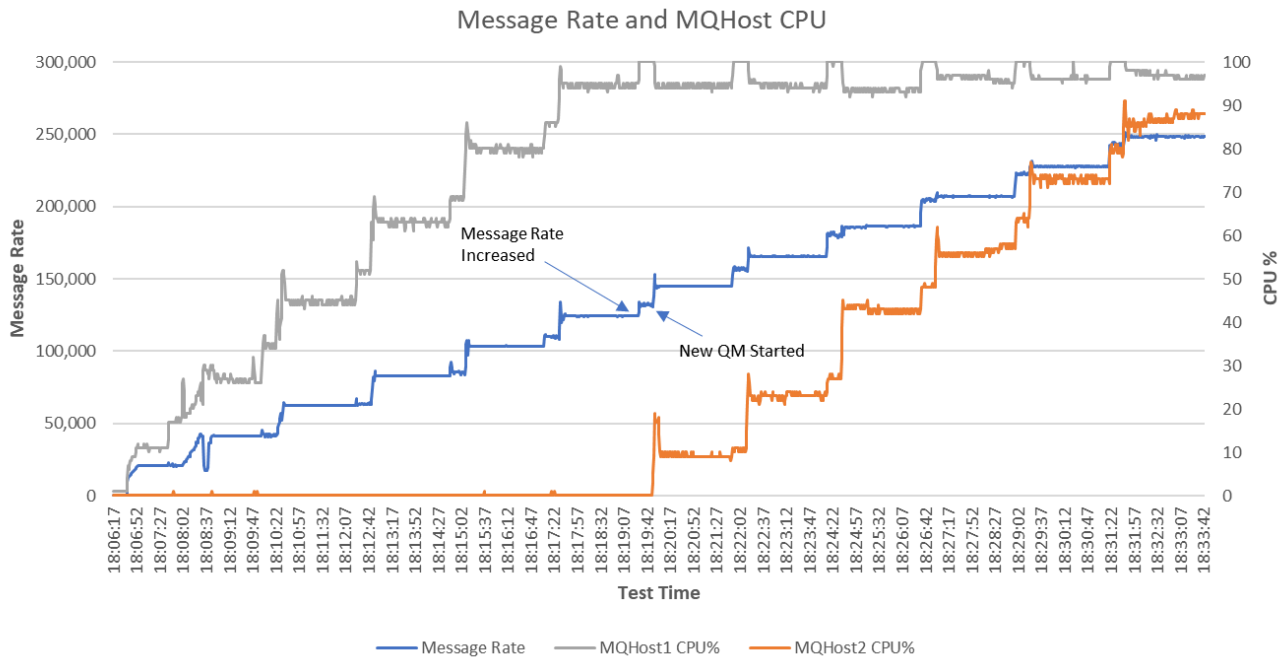


FIGURE 6: MESSAGE RATE AND MQHOST CPU%

Figure 7 shows the cgroup slice CPU (400% = 4 cores) of a selected number of QMs (UC01QM01,UC01QM02, UC01QM07 & UC01QM08) and the number of receivers connected to the first QM (UC01QM01).

When the rate is incremented for the 2nd time (towards the start of the test) UC01QM01 reaches its slice limit of 400% until the 2nd QM is started. Initial QMs on a machine tend to consume less CPU than subsequent QMs, probably due to the lesser impact of context switching across the machine. CPU consumption of QMs on MQHost2 (starting with UC01QM07 & UC01QM08, shown) start out at a lower consumption once again, though lower still than the initial QMs on MQHost1.

The numbers of requesters shown connected to UC01QM01 is representative of the number of requesters on *any* active QM at that point in the test. The number of connected receivers per QM will be similar, so only the one plot is shown for clarity.

Note that as the number of applications on each QM approach the same number, re-balancing takes longer, to avoid continuously moving application to achieve a perfect balance, so at the end of the test QMHost2 has some QMs that are hosting slightly less requesters than QMHost2. If the test were allowed to continue, the uniform cluster function would perfectly balance across all twelve QMs. In production there will probably be applications connecting and disconnecting continuously, so constant re-balancing when the number of applications on each QM is very close already is not desirable.

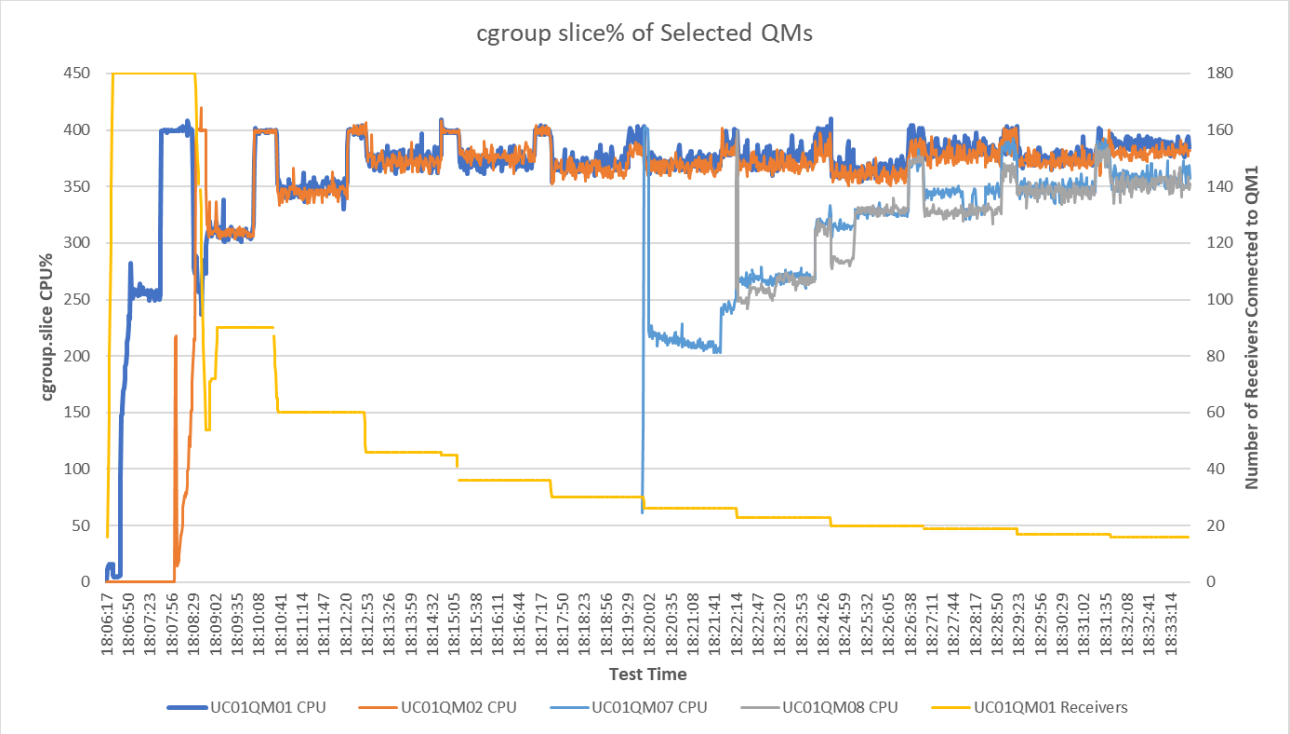


FIGURE 7: CGROUP CPU% AND RECEIVERS CONNECTED TO UC01QM01

1.2.4 Comparisons with Alternative Scenarios

In addition to the test above, peak rates and CPU were measured for the following scenarios:

Fixed High Rate (1-12QMs) 180 Senders and Receivers were started, connecting to a single, active QM (UC01QM). Senders were set to deliver 248,400 messages per second from the start. Additional QMs were started up to a total of 12, to take up the load.

Fixed High Rate (12QMs) 180 Senders and Receivers were started, connecting to 12 active QMs (UC01QM-UC01QM12). Senders were set to deliver 248,400 messages per second from the start. In this case the queue manager groups functionality will distribute the connections across the 12 QMs from the start.

Fixed High Rate (2 QMs) 180 Senders and Receivers were started, connecting to 2 unrestricted, active QMs (UC01QM01 & UC01QM07) outside of a cgroup slice, enabling the single QM on each host to consume all the host's CPU resources. Senders were set to deliver 248,400 messages per second from the start. In this case the queue manager groups functionality will distribute the connections across the 2 QMs from the start.

Test	Rate Achieved	MQHost1 CPU%	MQHost2 CPU%
Base Test	248,338	97	87
Fixed High Rate (1-12QMs)	248,300	94	90
Fixed High Rate (12QMs)	248,400	90	90
Fixed High Rate (2 unrestricted QMs)	248,100	96	95

Table 2

Table 2 above shows that with 12 uniform cluster QMs performance was similar. The load was slightly more balanced at the end, when the clients were initially connected across 12 active QMs, but over time the balancing logic of uniform cluster would expect to settle for the other cases as well.

For 2 large QMs the achieved rate was slightly less, thus, in addition to being able to only start QMs as they are needed, (with the associated saving in resources such as memory), splitting the load across multiple smaller QM's can achieve better throughout.

In production workloads will be different to these scenarios, probably combining surges of rates and possibly client connections at peak times. A variety of performance metrics can be monitored to establish trigger points for starting additional cluster members to take up the load (e.g. CPU usage, queue depth etc). Existing applications will be balanced across the cluster, as it expands, whilst newly started applications can use queue manager groups to spread connections across the cluster members that are active at any point. Similarly a uniform cluster can be contracted as demand decreases by issuing 'endmqm -r <QM>' which will move applications off the QM being stopped, onto the remaining active cluster members.

1.2.5 Queue Manager Setup

The uniform cluster configuration utilises the -ii, -ic and -iv flags of the crtmqm command, allowing every queue manager to use exactly the same configuration which makes horizontal scaling very easy. So, to create the first QM (UC01QM01), which is also repository one of the uniform cluster:

```
crtmqm -lc -p 1414 -ii /nfs1/uniclus.ini -ic /nfs1/uniclus.mqsc -iv
CONNAME=MQHost1(1414) UC01QM01
```

The -ii parameter points to a file of qm.ini attributes which is used to add or modify entries for the QM being created. For uniform cluster it must at least contain the essential entries defining the cluster (see below). It can also contain additional entries, applied to any QM created with it, like enabling FastPath channels, as in the example below. The same file can be used for all queue managers, so could be located on an NFS file system for access by QM's being created on different hosts for example.

```
AutoCluster:
Repository1Conname=MQhost1(1414)
Repository1Name=UC01QM01
Repository2Conname=MQhost2(1414)
Repository2Name=UC01QM07
ClusterName=UC01
Type=Uniform
#
# Custom entries
Channels:
   MQIBindType=FASTPATH
```

Sample uniform cluster ini file used by the -ii parameter

The -ic parameter points to an mqsc command file that is run by the queue manager created with this, on every start-up. It must contain at least the cluster receiver channel for the QM and can make use of in-built variables and variables specified on the crtmqm command (-iv parameter).

```
# Uniform cluster receiver channel
define channel('+AUTOCL+_+QMNAME+') chltype(clusrcvr) trptype(tcp)
conname('+CONNAME+') cluster('+AUTOCL+') replace

# Uniform cluster receiver channel
alter LISTENER(SYSTEM.LISTENER.TCP.1) TRPTYPE(TCP) BACKLOG(5000)
define channel(PERF.APP.CHL) chltype(svrconn) trptype(tcp) replace
alter channel(PERF.APP.CHL) chltype(SVRCONN) sharecnv(1)
define qlocal(request1) replace
define qlocal(request2) replace
…
```

Sample mqsc file used by the -ic parameter of crtmqm

In the sample file above, the mqsc file makes use of the +AUTOCL+ & +QMNAME+ inbuilt variables and the +CONNAME+ variable specified on the command line to create the cluster receiver channel

When the -ii and -ic files are set up in this way, very little needs to be changed to create multiple cluster members. In this example we merely change the QM name, host and listener port on the crtmqm command. E.g. to create QMs UC01QM01 to UC01QM03 on QMHost1 and QMs UC01QM07 to UC01QM09 on QMHost2:

On QMHost1 execute:

```
crtmqm -lc -p 1414 -ii /nfs1/uniclus.ini -ic /nfs1/uniclus.mqsc -iv
CONNAME=MQHost1(1414) UC01QM01

crtmqm -lc -p 1415 -ii /nfs1/uniclus.ini -ic /nfs1/uniclus.mqsc -iv
CONNAME=MQHost1(1415) UC01QM02

crtmqm -lc -p 1416 -ii /nfs1/uniclus.ini -ic /nfs1/uniclus.mqsc -iv
CONNAME=MQHost1(1416) UC01QM03
```

On QMHost2 execute:

```
crtmqm -lc -p 1414 -ii /nfs1/uniclus.ini -ic /nfs1/uniclus.mqsc -iv
CONNAME=MQHost2(1414) UC01QM07

crtmqm -lc -p 1415 -ii /nfs1/uniclus.ini -ic /nfs1/uniclus.mqsc -iv
CONNAME=MQHost2(1415) UC01QM08

crtmqm -lc -p 1416 -ii /nfs1/uniclus.ini -ic /nfs1/uniclus.mqsc -iv
CONNAME=MQHost2(1416) UC01QM09
```

1.2.6 Machines used in the Test

QM_host1 ThinkSystem SR630
CPU 2 x 12: Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz

QM_host2 ThinkSystem SR630
CPU 2 x 12: Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz

App_host1 System x3550 M5
CPU 2 x 14 Intel(R) Xeon(R) E5-2690 v4 @ 2.60GHz

App_host2 System x3550 M5
CPU 2 x 14 Intel(R) Xeon(R) E5-2690 v4 @ 2.60GHz

1.2.7 PerfHarness Commands

For the main test above, the following PerfHarness commands were used, should you wish to run something similar. Note that you will need an up to date version of PerfHarness from Github (link below) to pick up some of the recent changes to support uniform cluster testing. The SocketCommandProcessor class was used to control the rate of the senders.

Receivers:

java -Xms768M -Xmx768M -Xmn600M JMSPerfHarness -su -wt 10 -wi 0 -nt 180 -ss 5 -sc BasicStats -rl 0 -id 5 -tc jms.r11.Receiver -d REQUEST -to 20 -db 1 -dx 1 -dn 1 -jb *UC01QM -jt mqc -pc WebSphereMQ -ccdt file:///nfs1/UC01.json -ar WMQ_CLIENT_RECONNECT -an mqperf.receiver -wp true -wc 4

Senders:

java -Xms768M -Xmx768M -Xmn600M JMSPerfHarness -su -wt 10 -wi 0 -nt 180 -ss 5 -sc BasicStats -rl 0 -id 5 -tc jms.r11.Sender -d REQUEST -to 20 -db 1 -dx 1 -dn 1 -jb *UC01QM -jt mqc -pc WebSphereMQ -ms 2048 -rt 1 -ccdt file:///nfs1/UC01.json -ar WMQ_CLIENT_RECONNECT -an mqperf.senders -cmd_c SocketCommandProcessor

2 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-CB	Client mode requesters on separate host. Binding mode responders.
RR-DQ-BB	Distributed queueing between two queue managers on separate hosts, with binding mode requesters and responders.
RR-BB	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).

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.

2.1 RR-CB Workload (Client mode requesters on separate host. Binding mode responders.)

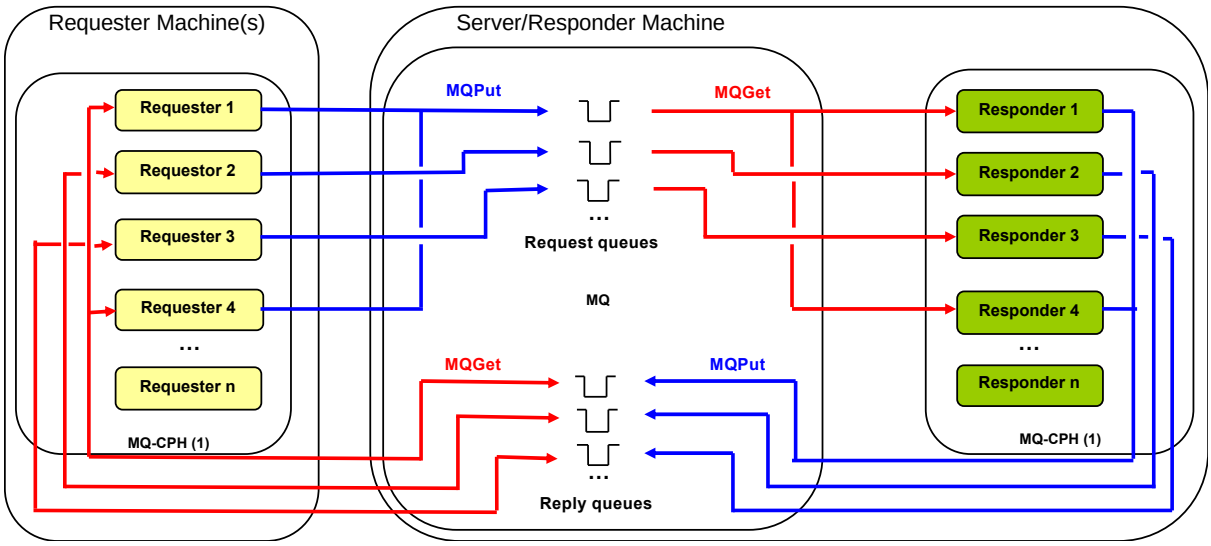


FIGURE 8 - REQUESTER-RESPONDER WITH REMOTE QUEUE MANAGER (LOCAL RESPONDERS)

Figure 8 shows the topology of the RR-CB 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. Additional machines may be used to drive the requester applications where necessary. 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, similar to the responders. The number of responders is set such that there is always a waiting 'getter' for the request queue.

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

Variants of the RR-CB test differ in the location of the applications (RR-BB & RR-CC).

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

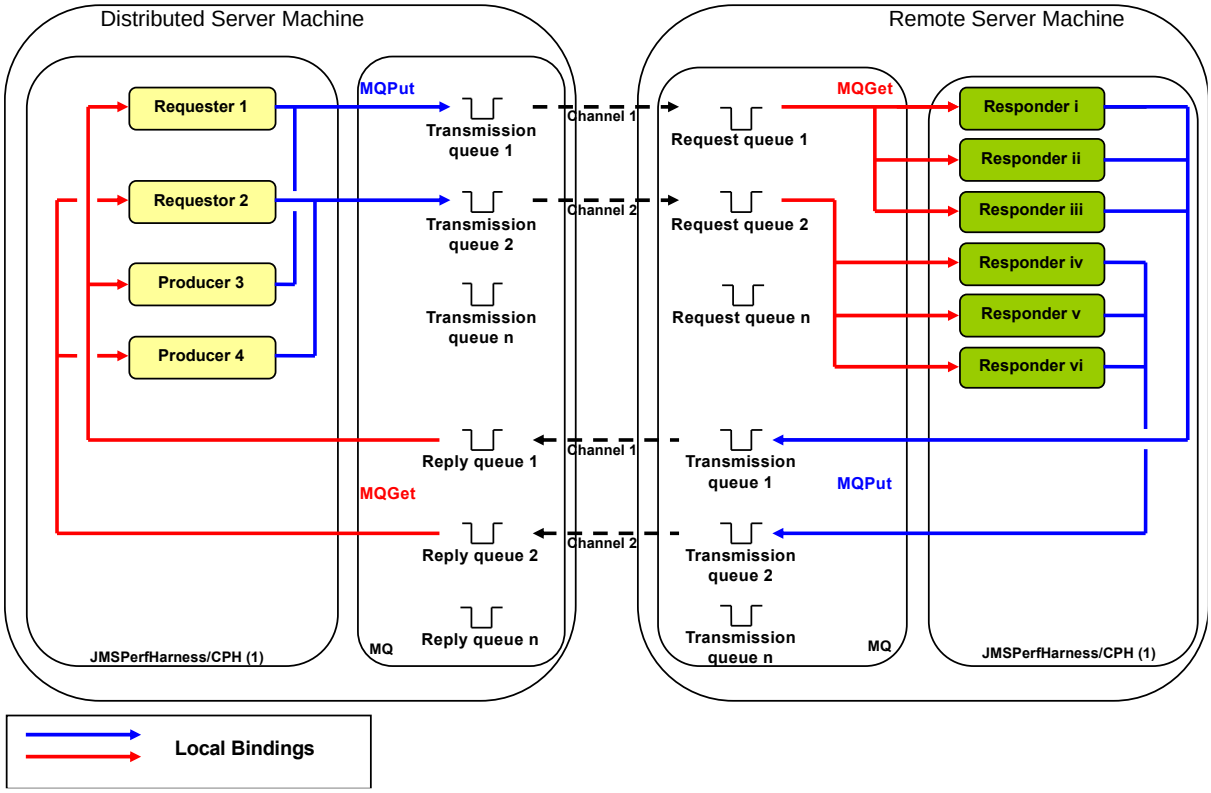


FIGURE 9 - REQUESTER-RESPONDER WITH REMOTE QUEUE MANAGER (REMOTE RESPONDERS).

This is a distributed queuing version of the requester-responder topology detailed in section 2.1. All *MQPUT*s are to remote queues so that messages are now transported across server channels to the queue manager where the queue is hosted.

3 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).

3.1 RR-CB Workload

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

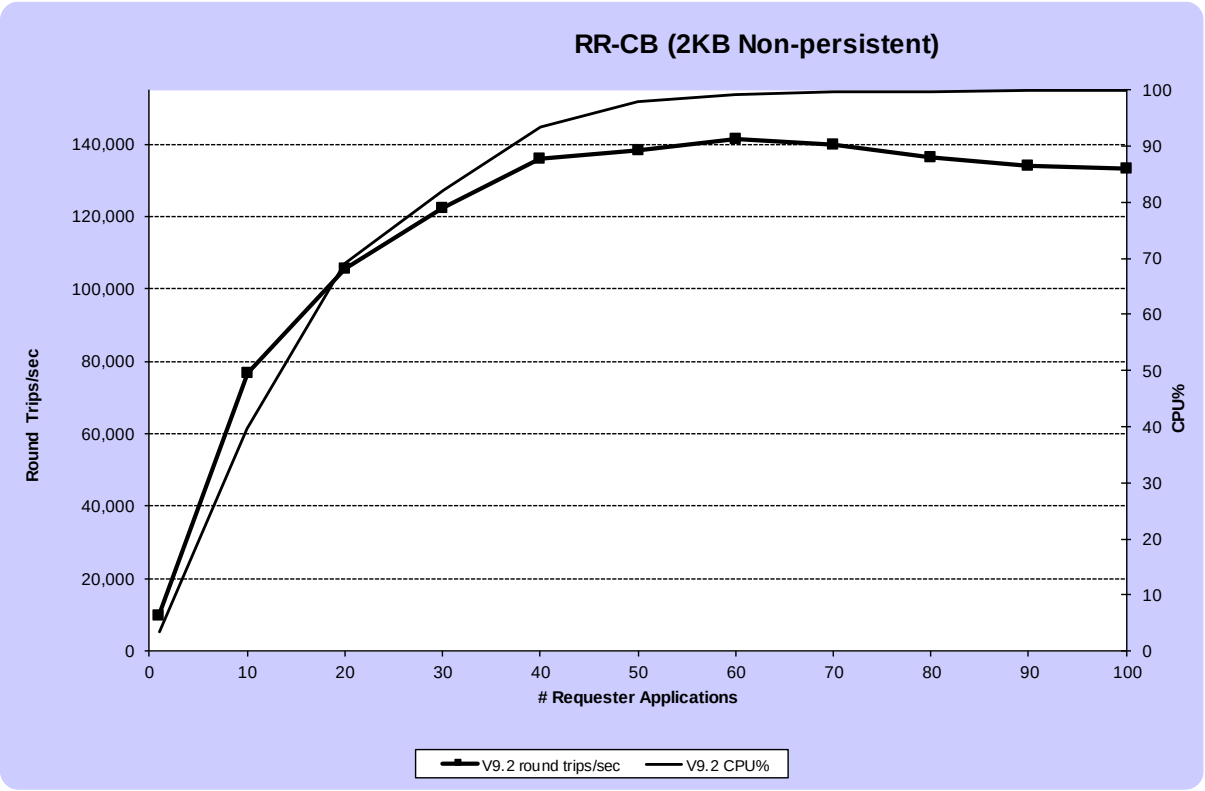


FIGURE 10 - PERFORMANCE RESULTS FOR RR-CB (2KB NON-PERSISTENT)

The test peaked at approximately 142,000 round trips/sec, fully utilising the CPU 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 being limited by the 40Gb network that links the client and server machines.

Test	V9.2 Max Rate*	CPU%	Clients
RR-CB (2KB Non-persistent)	141,554	99.27	60
RR-CB (20KB Non-persistent)	114,930	99.23	60
RR-CB (200KB Non-persistent)	22,639	57.39	50
RR-CB (2MB Non-persistent)	2,194	63.01	50

***ROUND TRIPS/ SEC**

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

3.1.1 Test setup

Workload type: RR-CB (see section 2.1).

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

3.2 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 0) 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 of the tests in this section utilise multiple send/receive channels. This particularly helps with smaller, non-persistent messages when the network is under-utilised.

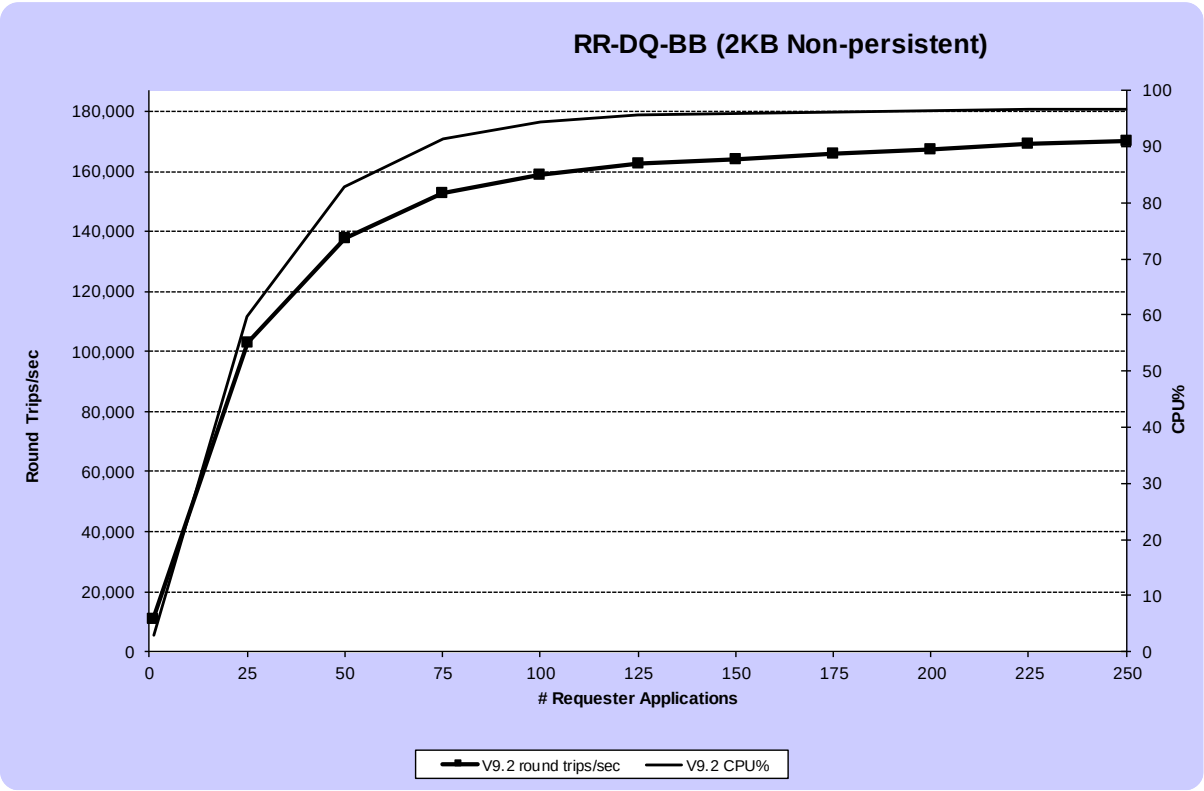


FIGURE 11 - PERFORMANCE RESULTS FOR RR-DQ-BB (2KB NON-PERSISTENT)

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 40Gb network.

Test	V9.2 Max Rate*	CPU%	Clients
RR-DQ-BB (2KB Non-persistent)	170,191	96.57	250
RR-DQ-BB (20KB Non-persistent)	121,909	91.82	120
RR-DQ-BB (200KB Non-persistent)	22,653	48	30
RR-DQ-BB (2MB Non-persistent)	2,161	53.57	25

***ROUND TRIPS/ SEC**

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

3.2.1 Test setup

Workload type: RR-DQ-BB (see section 0).

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

3.3 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).

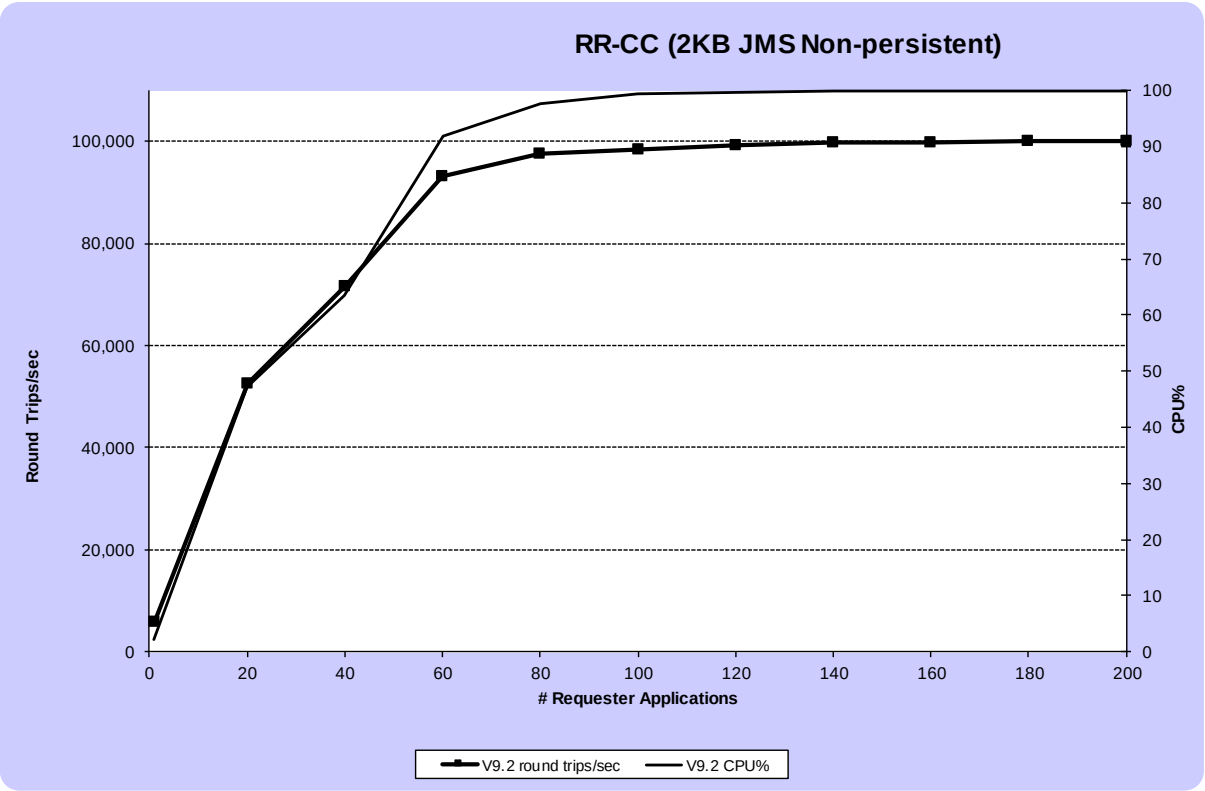


FIGURE 12 - PERFORMANCE RESULTS FOR RR-CC (2KB JMS NON-PERSISTENT)

Once again, the workload exhibits good scaling up to 100% of the CPU (the limiting factor), peaking at approximately 95,000 round trips/sec

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 40Gb network; the rates are lower than the RR-CB network limited scenarios because of the additional network hop to the responder applications which are local in the RR-CB scenario.

Test	V9.2 Max Rate*	CPU%	Clients
RR-CC (2KB JMS Non-persistent)	100,136	99.99	180
RR-CC (20KB JMS Non-persistent)	81,072	99.96	150
RR-CC (200KB JMS Non-persistent)	10,912	57.29	100
RR-CC (2MB JMS Non-persistent)	1,011	52.47	90

***ROUND TRIPS/ SEC**

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

3.3.1 Test setup

Workload type: RR-CC (see section 2).

Message protocol: JMS

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

3.4 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 6 strongest TLS1.2 MQ CipherSpecs, and all TLS1.3 MQ CipherSpecs (support for TLS 1.3 encryption was introduced in MQ V9.1.4).

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

- TLS_RSA_WITH_AES_256_CBC_SHA256
- TLS_RSA_WITH_AES_256_GCM_SHA384
- 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.

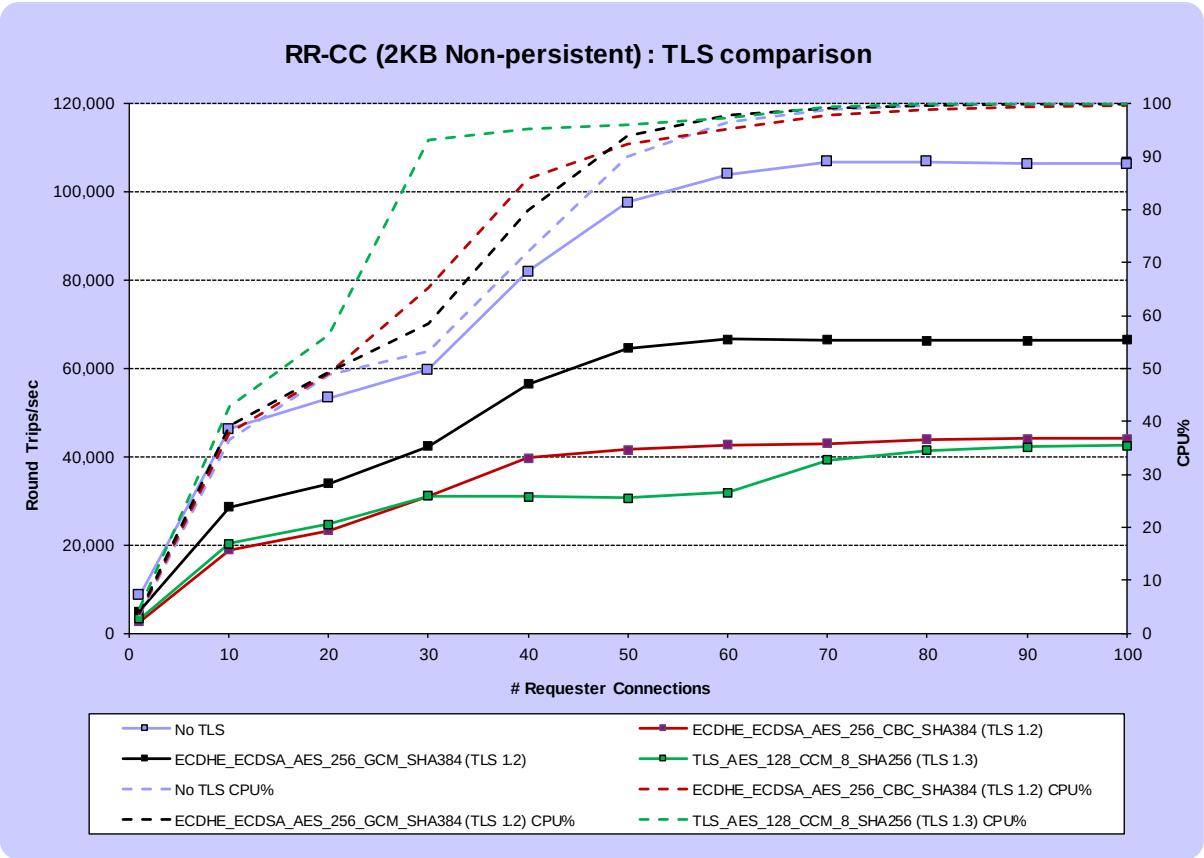


FIGURE 13 - PERFORMANCE RESULTS FOR RR-CC WITH TLS 1.2

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 of 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 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 39% of a non-encrypted workload. TLS 1.3 encryption is more expensive, acheiving rates slightly below the TLS 1.2 CBC based CipherSpecs.

TLS 1.2 CipherSpec	V9.2 GM		
	Max Rate*	CPU%	Clients
No TLS	106,826	100	80
TLS_RSA_WITH_AES_256_CBC_SHA256	43,914	100	100
TLS_RSA_WITH_AES_256_GCM_SHA384	66,592	98	60
ECDHE_ECDSA_AES_256_CBC_SHA384	44,132	100	100
ECDHE_ECDSA_AES_256_GCM_SHA384	66,594	98	60
ECDHE_RSA_AES_256_CBC_SHA384	43,893	100	100
ECDHE_RSA_AES_256_GCM_SHA384	66,626	99	70

**Round trips/ sec*

Table 6 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	V9.2 GM		
	Max Rate*	CPU%	Clients
No TLS	106,826	100	80
TLS_RSA_WITH_AES_256_CBC_SHA256	43,914	100	100
TLS_RSA_WITH_AES_256_GCM_SHA384	66,592	98	60
ECDHE_ECDSA_AES_256_CBC_SHA384	44,132	100	100
ECDHE_ECDSA_AES_256_GCM_SHA384	66,594	98	60
ECDHE_RSA_AES_256_CBC_SHA384	43,893	100	100
ECDHE_RSA_AES_256_GCM_SHA384	66,626	99	70

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	V9.2 GM		
	Max Rate*	CPU%	Clients
No TLS	106,826	100	80
TLS_AES_128_CCM_8_SHA256	42,539	100	100
TLS_AES_256_GCM_SHA384	44,237	100	100
TLS_CHACHA20_POLY1305_SHA256	43,079	100	100
TLS_AES_128_GCM_SHA256	44,165	100	100
TLS_AES_128_CCM_SHA256	42,809	100	100

Round trips/ sec

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

3.4.1 Test setup

Workload type: RR-CC (see section 2).

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

4 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, and then results for V9.2 in a separate environment (x64 Linux with SAN, SSD & NFS filesystems) are shown to demonstrate the impact of transaction log location.

4.1 RR-BB Workload

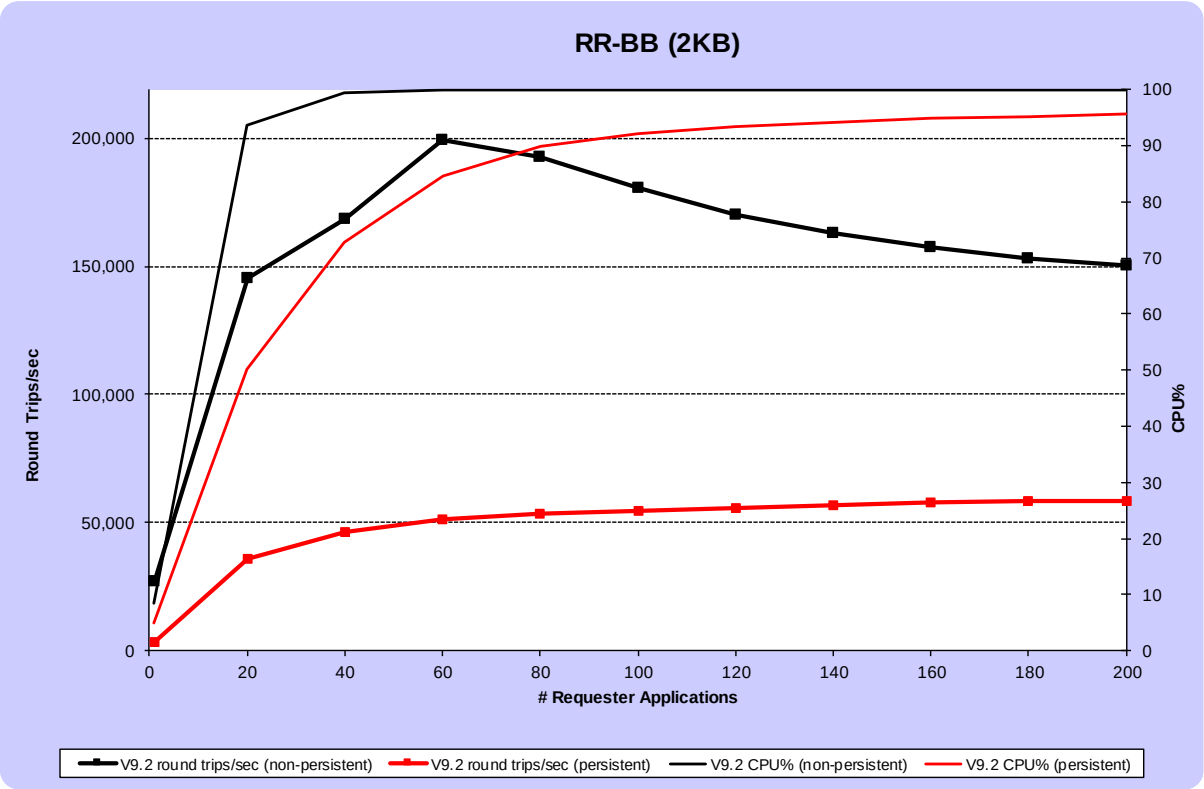


FIGURE 14 - PERFORMANCE RESULTS FOR RR-BB (2KB NON-PERSISTENT VS PERSISTENT)

Figure 14 shows results from running the RR-BB workload with 2KB non-persistent and persistent messages, on the same server used for the non-persistent scenarios in the previous sections.

RR-BB is a variant of RR-CB (see section 2.1) where all applications are connected in bindings mode. This accentuates the impact of persistent messaging since we are no longer limited by network bandwidth.

The non-persistent workload reaches an optimal value at 60 requesters where, the CPU reaches 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 component of IBM MQ to aggregate log data into larger, more efficient writes to disk.

Peak round trip rates for all message sizes tested, for persistent & non-persistent scenarios can be seen in the tables below.

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, 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.

Test	V9.2 Max Rate*	CPU%	Clients
RR-BB (2K Non-persistent)	199,421	99.99	60
RR-BB (20K Non-persistent)	130,598	99.62	50
RR-BB (200K Non-persistent)	52,270	93.04	27
RR-BB (2MB Non-persistent)	3,377	80.12	20

***ROUND TRIPS/ SEC**

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

Test	V9.2 Max Rate*	CPU%	Clients
RR-BB (2KB Persistent)	58,515	95.76	200
RR-BB (20KB Persistent)	41,615	79.68	100
RR-BB (200KB Persistent)	6,912	25.99	20
RR-BB (2MB Persistent)	744	21.17	10

***ROUND TRIPS/ SEC**

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

4.1.1 Test setup

Workload type: RR-BB (see section 2).

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

4.2 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 transaction logs.

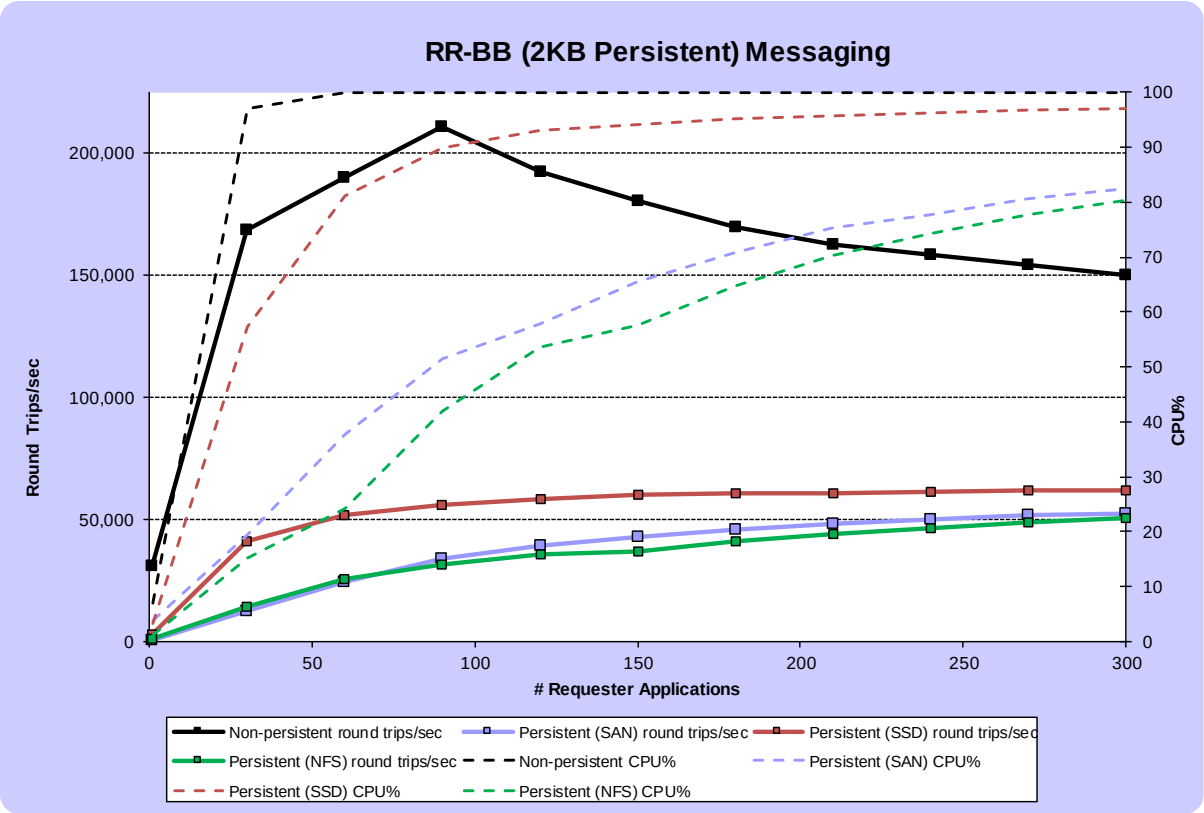


FIGURE 15 - PERFORMANCE RESULTS FOR RR-BB PERSISTENT MESSAGING LOGGING TO SSD, SAN & NFS

To illustrate the impact that the filesystem hosting the transaction logs can have, Figure 15 shows results from running the RR-BB workload with non-persistent and persistent messaging (hosted on different filesystems).

RR-BB is a variant of RR-CB (see section 2.1) where all applications are connected in bindings mode, eliminating the network as a bottleneck (except in the case of NFS, where there is a 10Gb link from the MQ server to the NFS server).

As expected, the non-persistent case is limited by CPU

1.1.1 Test setup

Workload type: RR-BB (see section 2).

Hardware: x64 Linux MQ server, x64 Linux NFS server, IBM SAN Volume Controller, and Storwise V7000 – see section A.2.

Appendix A: Test Configurations

A.1 Hardware/Software – Set1

All of 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:

Lenovo System x3550 M5 – [5463-L2G]

2 x 12 core CPUs.
Core: Intel® Xeon® E5-2690 v3 @ 2.60GHz

128GB RAM

40Gb ethernet adapters connect all three machines via an isolated performance LAN.

A.1.2 Software

Red Hat Enterprise Linux Server release 7.7 (Maipo)

JMSPerfHarness test driver (see Appendix C:)

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

IBM MQ V9.2

A.2 Hardware/Software – Set2 (Persistent messaging comparisons)

The persistent messaging tests in section 4.2, comparing different filesystems used to host the MQ transaction logs, were run with the following hardware/software.

A.2.1 Hardware

The MQ Server and NFS Server are two identical machines:

Lenovo System x3550 M5 – [8869-AC1]

2 x 14 core CPUs.
Core: Intel® Xeon® E5-2690 v4 @ 2.60GHz

128GB RAM

10Gb ethernet adapters connected the QM server to the NFS server, via an isolated performance LAN.

The SAN test used an IBM Storwize V7000 populated with 10,000 rpm disks configured in a RAID 10 array, and fronted by an IBM SAN Volume Controller (SVC) with 20GB of RAM. The SVC was connected to the MQ server via a dual-port 8Gb fibre channel adapter.

A.2.2 Software

Red Hat Enterprise Linux Server release 7.7 (Maipo)

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

IBM MQ V9.2

A.3 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.3.1 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 = 13121479
net.ipv4.ip_local_port_range = 1024 65535
vm.max_map_count = 1966080
kernel.pid_max = 655360
kernel.sem = 1000 1024000 500 8192
kernel.msgmnb = 131072
kernel.msgmax = 131072
kernel.msgmni = 32768
kernel.shmmni = 8192
kernel.shmall = 4294967296
kernel.shmmax = 137438953472
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
@mqm soft core unlimited
@mqm hard core unlimited
```

A.3.2 IBM MQ

The following parameters are added or modified in the qm.ini files for the tests run in section 3 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.

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/

Queue Manager Restart Time Improvements in V9.1.1 Performance Paper
https://ibm-messaging.github.io/mqperf/Queue%20Manager%20Restart%20Times.pdf

Improved Switch/Fail-over times in MQ V9.1.2 Blog Article
https://community.ibm.com/community/user/imwuc/viewdocument/improved-switchfail-over-times-in

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

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

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