IBM MQ V9.3.3 for Linux (x86-64 platform)

AMQP JMS Performance Report

Version 1.0 - September 2023

Paul Harris

IBM MQ Performance

Hursley Park, UK

Savitha Joshi

XMS/AMQP Development

Bangalore, India

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, July 2023.

This edition applies to *IBM MQ V9.3.3* (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 AMQP support in IBM MQ V9.3.3 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.

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 AMQP JMS in IBM MQ v9.3.3 for Linux on x86_64.
- Want to understand the performance limits of AMQP JMS in IBM MQ v9.3.3 for Linux on x86_64.
- Want to understand what actions may be taken to tune support for AMQP JMS in IBM MQ v9.3.3 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:

- Examples of AMQP JMS messaging scenarios, with performance results on specified hardware.
- Tuning advice specific to AMQP JMS support in MQ V9.3.3

Feedback

We welcome feedback on this report.

- Does it provide the sort of information you want?
- Do you feel something important is missing?
- Is there too much technical detail, or not enough?
- Could the material be presented in a more useful manner?

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

Please direct any feedback on this report to paul_harris@uk.ibm.com or jsavitha@in.ibm.com

Contents

TABLES

FIGURES

1 Introduction

IBM MQ supports applications connecting over multiple network protocols. This includes its own MQ protocol and the open standard AMQP 1.0 network protocol.

Both of these protocols support many messaging capabilities, and both provide routes into MQ for Java applications using the open standard API, JMS. The MQ protocol is highly optimised for particularly high volume MQ workloads. AMQP provides the benefit to applications of allowing the use of a vendor-neutral, open source, client, such as Apache Qpid JMS.

This report will focus on various performance characteristics when using the AMQP protocol, first highlighting the performance improvements in the IBM MQ 9.3.3 release over previous releases, and then a comparison between using the AMQP protocol and JMS client and the MQ protocol and JMS client.

Note that some of the results in this report are dependent on the fix to APAR IT44457 (https://www.ibm.com/support/pages/apar/IT44457).

2 Performance characteristic

Performance characteristics in AMQP refer to various aspects that determine the reliability, durability, acknowledgement mode, message size and infrastructure. Here are some key performance characteristics to consider:

2.1 Durability

The durability of message is ability to persist and survive even in the case of system failures, crashes, or restarts. ensures that important data is not lost due to temporary outages or unexpected events. Message can be persistent or non-persistent.

2.1.1 Persistent

Persistent messages are written to durable storage(such as a disk) before it is delivered. This ensures that the message survives even in case of failure or restart. Storing messages on disk introduces additional I/O overhead, which can impact performance compared to non-persistent messages. So Persistent messages typically should be used mainly for critical data that must not be lost.

2.1.2 Non-Persistent

Non-persistent messages are not stored in a durable manner. They will be kept in memory or temporary storage. So, these messages are not guaranteed to survive system failures. Non-persistent messages generally offer better performance as they don't need to be stored on disk, reducing I/O operations and improving throughput. Non-persistent messages are suitable for scenarios where immediate delivery is more important than durability, and losing the message in the event of a failure is acceptable.

2.2 Acknowledge Mode

Acknowledge mode refers to how and if consumer acknowledges the receipt and processing of messages from a queue or topic. Acknowledgements enable the server to determine whether the message was successfully delivered across the network or not, and therefore, whether the message can be correctly removed or must remain for redelivery. There are different acknowledge modes:

2.2.1 Acknowledge mode

In this mode, messages are delivered at-least-once delivery, this is achieved through message acknowledgment, redelivery mechanisms and use of persistent messages. If a message delivery fails or the consumer fails to acknowledge the message, the message will be redelivered until successful acknowledgment is received. This will ensure messages are not lost **but may result in duplicate messages being delivered.** Using acknowledgment mode adds some overhead as the application needs to send acknowledgment messages back to the AMQP service. This introduces additional network communication and processing time.

2.2.2 No acknowledge mode

In this mode, AMQP provides at-most-once delivery where a message is delivered once and is not redelivered in case of failures. This QoS level is suitable for non-critical messages where occasional message loss is acceptable.

To enable at-most-once behaviour with the Apache Qpid JMS client, extended session acknowledgement mode of 'No Acknowledge' on the JMS session at creation (referred to as NoAck in the report) to support At-Most-Once.

"In this mode messages are accepted at the server before being dispatched to the client, and no acknowledgement is performed by the client." - https://qpid.apache.org/releases/qpid-jms-2.2.0/docs/index.html

No acknowledgment mode eliminates the need for explicit acknowledgments, potentially reducing some overhead. However, it also means that message loss is possible, as messages are considered delivered once sent to consumers.

2.3 Combination of Durability and Acknowledge Mode

You can use non-persistent messages in combination with acknowledgment modes "ack" or "noack."

2.3.1 Non-Persistent Messages with Acknowledgment Mode

In this scenario, the consumer would process the message, acknowledge its receipt, and then the message would be removed from the queue. However, since the message is non-

persistent, in case of failure or service crash occurs before the message is delivered, it might be lost.

2.3.2 Non-Persistent Messages with No Acknowledgment Mode

In this case, messages are delivered to consumers without requiring explicit acknowledgments. This mode is typically used when message loss is acceptable.

It's important to note that the combination of non-persistent messages and acknowledgment modes like "ack" or "noack" may result in message loss if service or consumer failure occurs.

2.3.3 General points to consider:-

When combining these factors, here are some general points to consider:-

For scenarios where high performance and immediate processing are crucial, non-persistent messages with no acknowledgment mode may be suitable.

When durability and reliability are paramount, persistent messages with acknowledgment mode should be used, even though there might be some impact on performance due to disk I/O and acknowledgment messages.

Balancing performance and durability may require experimenting and tuning based on your application's specific requirements.

It's important to benchmark and profile your application under different scenarios to determine the optimal combination of persistence and acknowledgment modes for your use case.

2.4 Comparison to MQ’s ack

The AMQP protocol does not provide the application with a confirmation that an acknowledged message has been removed. Therefore, the application must accept that a failure could result in the acknowledgement being lost, and one or more consumed messages being redelivered at a later point.

This is unlike IBM MQ’s protocol, where the consuming application will block until it knows that the acknowledgement has been successfully processed by the queue manager, and the message(s) removed. Therefore, higher assurances over consumption of messages can be provided with the MQ protocol.

3 AMQP Performance Improvements in MQV9.3.3

For noack mode i.e. in this mode, AMQP provides at-most-once delivery where a message is got from MQ and delivered once and is not redelivered in case of failures. Since there is no acknowledgment processing, it potentially reduces some overhead and performance is better compared to ack mode.

For ack mode , i.e. "at least once" behaviour of AMQP (Advanced Message Queuing Protocol), a message is guaranteed to be delivered to the consumer at least once. To achieve this, messages are delivered to the consuming application without removing them from the queue. Only once the consumer acknowledges receipt of the message, it is removed from the queue. If the application fails to acknowledge, the message is redelivered to the consumer to ensure successful processing. Since there is additional processing and network communication for acknowledged messages, there is higher overhead to process acknowledged message individually and impact on performance.

As the AMQP protocol does not provide the application with a confirmation that an acknowledged message has been removed, IBM MQ 9.3.3 has been enhanced to take advantage of this and to remove acknowledged messages in batches if consumed in quick succession, to improve the performance significantly (see Performance improvements for processing of AMQP message acknowledgments).

For ack mode, instead of processing acknowledge message individually, acknowledged messages are processed and removed in batch.

Processing messages in batches can increase overall throughput by allowing the system to handle multiple messages at once. This increases the performance significantly.

With MQ 9.3.3, All AMQP JMS workloads should show an improvement in comparison to MQ 9.3.0, particularly with smaller messages. As an example, Figure 1 below, shows a comparison between a requester/responder workload in MQ V9.3.0 and V9.3.3.

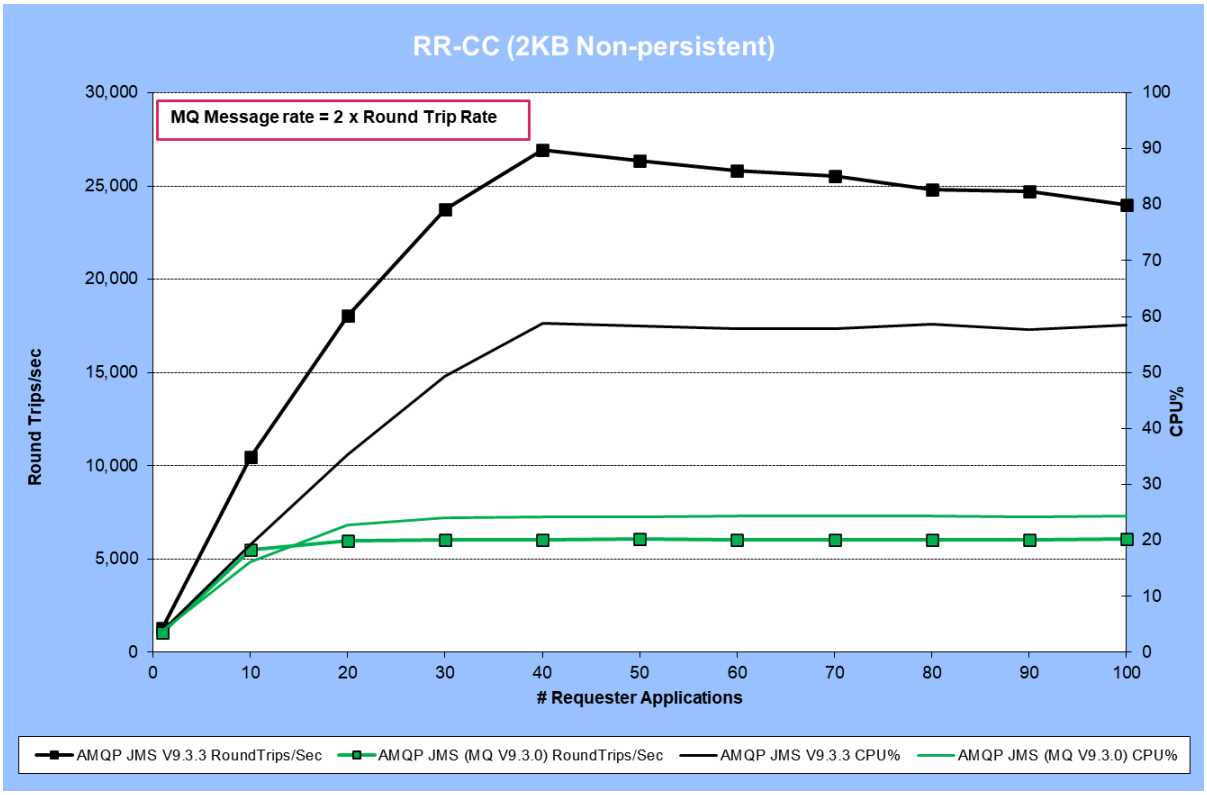


FIGURE 1 -AMQP JMS PERFORMANCE RESULTS FOR RR-CC (2KB NON-PERSISTENT) MQ V9.3. VS MQ V9.3.3

Workloads descriptions including 'RR-CC' shown above are described in the next section.

AMQP JMS configuration options for performance are detailed and discussed in section 9.

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

All measurements in this report were run against MQ V9.3.3 unless otherwise specified.

4 AMQP JMS Performance Workloads

Table 1 (below) lists the workloads used in the generation of performance data for the AMQP tests in this report. Three types of workload were run:

- Requester/responder (RR) scenarios
  These 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).

- Sender/receiver (SR) scenarios.
  These are asynchronous, point to point tests. A number of sender applications put messages on queues as fast as they can (one queue per sender), whilst receiver applications (at least one per queue) retrieve the message from the queues.

- Pub/Sub (PS) scenarios.
  These are asynchronous, Publish/Subscribe tests with a one or more publishers putting message on a topic unique to that publisher, and 1 or more subscribers per topic retrieve the messages

Workload	Description
RR-CC	Requester/responder with remote AMQP (Qpid) JMS requester applications , and remote Qpid JMS responder applications, all on separate, unique hosts.
SR-CC	Sender/receiver test with remote AMQP (Qpid) JMS sender applications , and remote Qpid JMS receiver applications, all on separate, unique hosts.
PS-CC	Publish/subscribe scenario with remote AMQP (Qpid) JMS publisher applications, and remote Qpid JMS subscriber applications, all on separate, unique hosts.

TABLE 1 - WORKLOAD TYPES

RR-CC, SR-CC and PS-CC are described in more detail in section 4.2 onwards.

4.1 Applications, Threads and Processes

From a queue manager's perspective in the workloads described below, each connection represents a unique application. The workloads are driven by the Perfharness client emulator tool. This tool is multi-threaded so 10 applications may be represented by 10 threads within a single Perfharness 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 PerfHarness, spread across as many processes as makes sense.

4.2 RR-CC (Requester/responder) Workload

(Client mode requesters on separate host. Client mode responders on separate host).

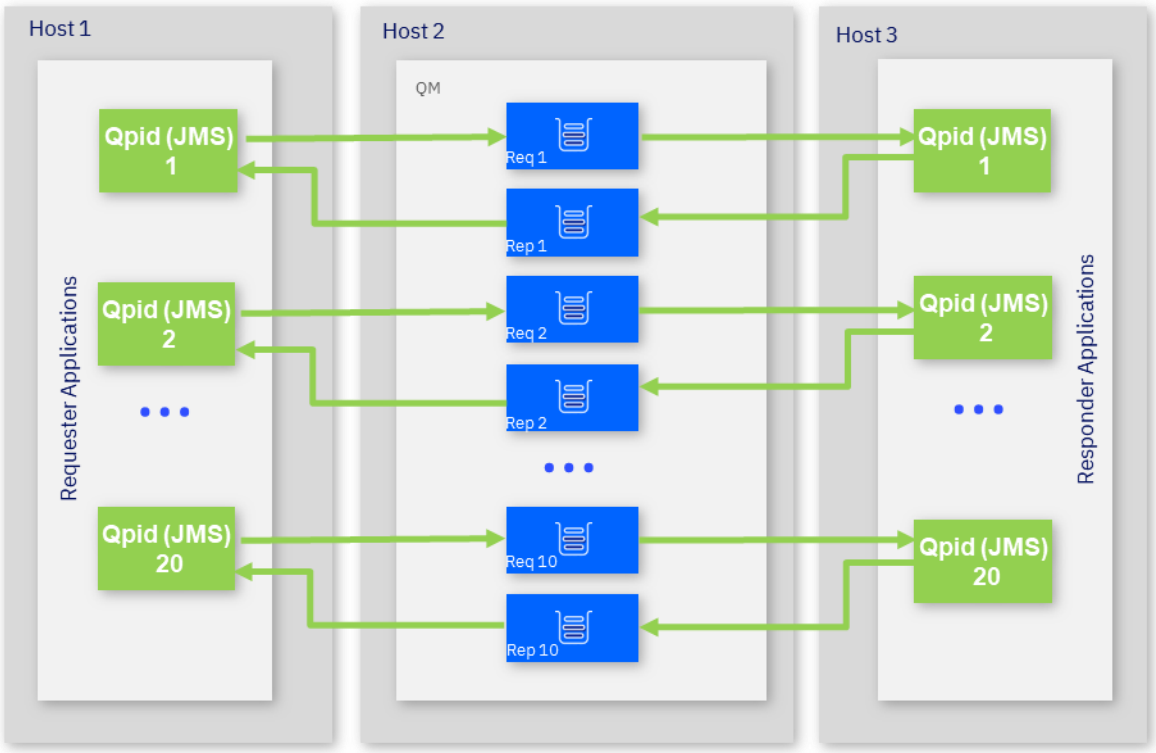


FIGURE 2 - REQUESTER-RESPONDER WITH REMOTE QUEUE MANAGER

Figure 2 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 Qpid(JMS) 11 will wrap round to use the Rep1/Req1 queues, and QPID (JMS) 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.

4.3 SR-CC (Sender/receiver) Workload

(Client mode senders on separate host. Client mode receivers on separate host).

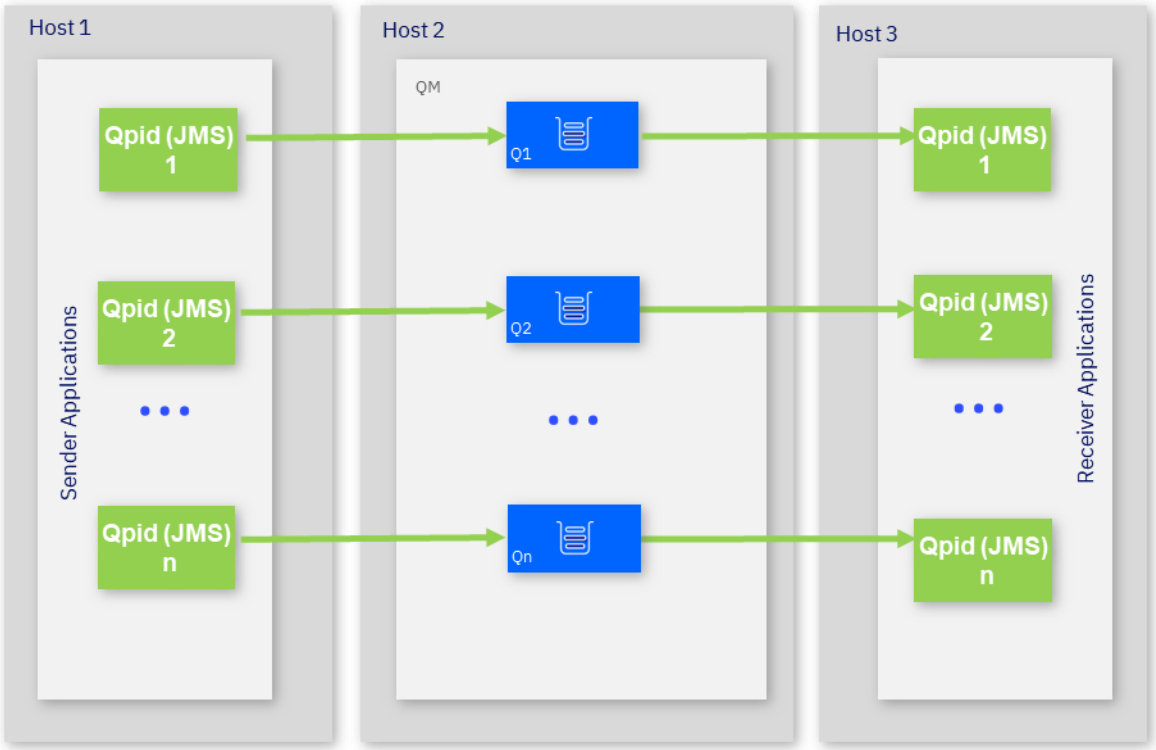


FIGURE 3 - SENDER-RECEIVER WITH REMOTE QUEUE MANAGER.

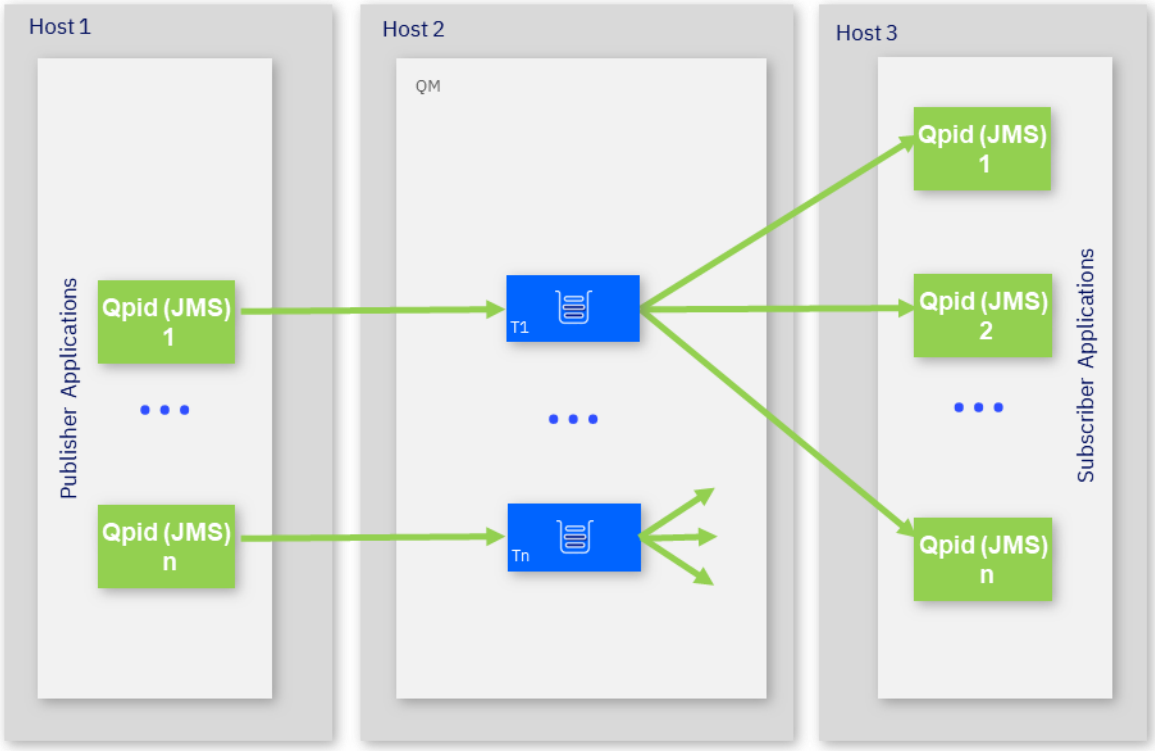
This scenario is a simpler point-to-point messaging test with messages being sent to queues from client mode sender applications and consumed by client mode receiver applications. Each sender/receiver application pair use their own dedicated queue.

Note that the nature of this test means that if the receiver applications can't consume messages as fast as the sender applications put messages on the queues, the queue can build up. Deep queues result in a different performance profile as messages may need to be written to the underlying queue file where they wouldn't if the queue was shallow.

The diagram above shows sender/queue/putter units (often referred to a s a 'triplet'), but all the tests in this report utilised 2 getters per queue to ensure there is always a getter available and minimise the possibility of queue filling. Having enough getter applications available to keep queues shallow is good practise in any scenario.

4.4 PS-CC (|Publish/subscribe) Workload

(Client mode publishers on separate host. - Client mode subscribers on separate host).



This scenario tests a Publish/Subscribe 'fan-out' environment. One or more Qpid JMS publisher applications puts messages on a topic that is unique to each publisher. Typically a single subscriber is started per topic and the test is called up by starting additional subscribers so that each additional published message is distributed to more and more subscribers as the test progresses.

4.5 Apache Qpid JMS (AMQP) vs IBM MQ JMS

The AMQP JMS client applications used in these tests are PerfHarness, utilising Qpid JMS (an inplementation of AMQP, exposed through the JMS api). Identical tests can be run with PerfHarness using the standard IBM MQ classes for JMS, and results for these are shown in this report as a comparison. Wherever the term IBM MQ JMS is used alone in this report, it refers to JMS using the IBM MQ JMS client which communicates over the IBM MQ protocol, whilst AMQP JMS refers to JMS using the Apache Qpid JMS client which communicated over the AMQP 1.0 protocol.

The IBM MQ JMS tests use client mode connections. fastpath channels and listeners (trusted) and have SHARECNV set to 1, which is the recommended value for performance.

Whilst the AMQP JMS and IBM MQ JMS tests look similar from the client API perspective the underlying function for each technology are siginificantly different, with a corresponding difference in performance.

JMS and AMQP service are both messaging protocols commonly used for building distributed applications. AMQP protocol(used by IBM AMQP Service) and MQ protocol (used by IBM MQ JMS). Decision between MQ Protocol or AMQP protocol depends on your specific needs and criteria. Refer Section 9 for more details on this.

Sections 7 and 8 show the relative performance of AMQP JMS vs IBM MQ JMS.

5 Non-Persistent Messaging Performance Test Results (AMQP JMS vs AMQP JMS with NoAck)

As said in introduction section, IBM MQ 9.3.3 release includes changes that improves performance mainly in acknowledge mode and no acknowledge mode during message delivery. Full performance test results for AMQP JMS are detailed below, including running with the no client acknowledgement 'NoAck' mode. The test results are presented by workload.

5.1 RR-CC (Requester/responder) Non-persistent Workload

(Client mode requesters on separate host. Client mode responders on separate host).

Figure 4, Figure 5 and Figure 6 below illustrate the performance of non-persistent messaging with various numbers of requester applications for 2KB, 20KB and 200KB message sizes.

A round-trip in a requester/responder (RR-CC) scenario, is composed of 2 MQPUTs and 2 MQGETs, so the underlying MQ message rate is two times the round-trip rate.

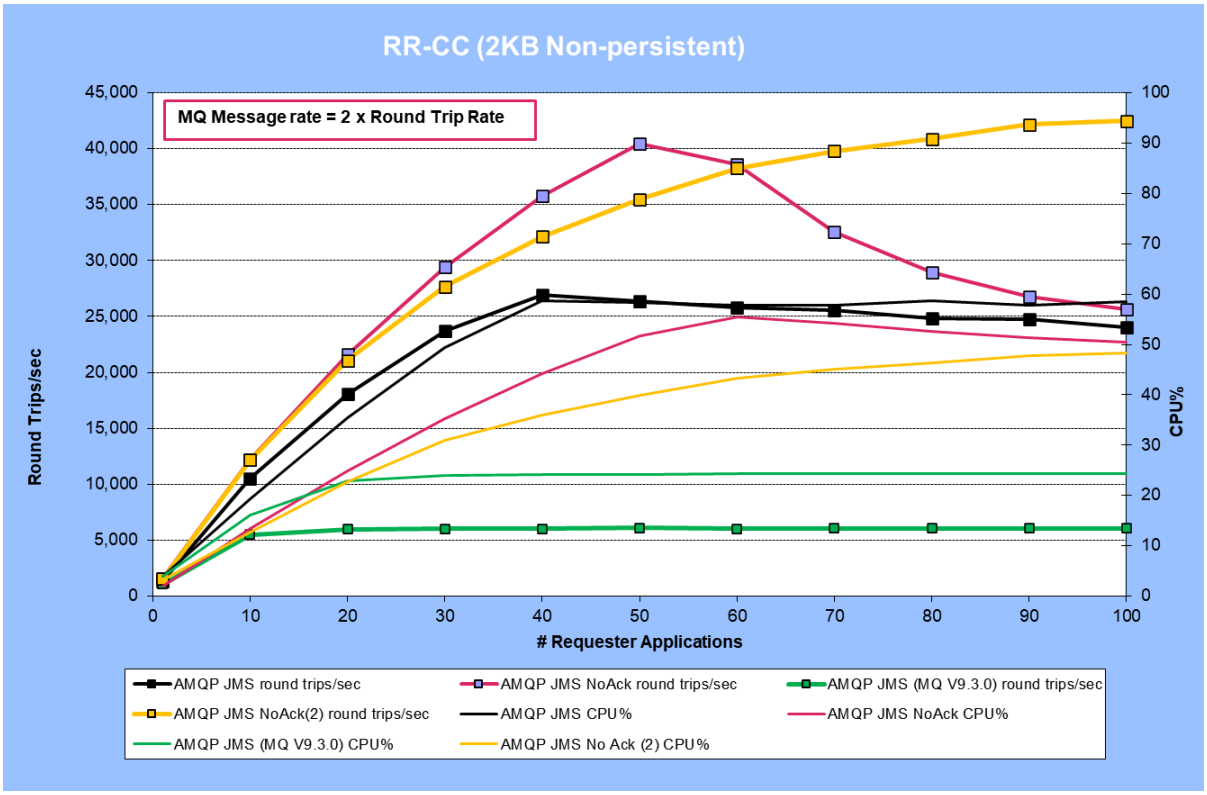


FIGURE 4 - RR-CC RESULTS (2KB NON-PERSISTENT, AMQP JMS VS AMQP JMS NOACK)

The "AMQP JMS NoAck (2)" plot in Figure 4 is an additional test with 32 AMQP worker threads set explicitly (see below).

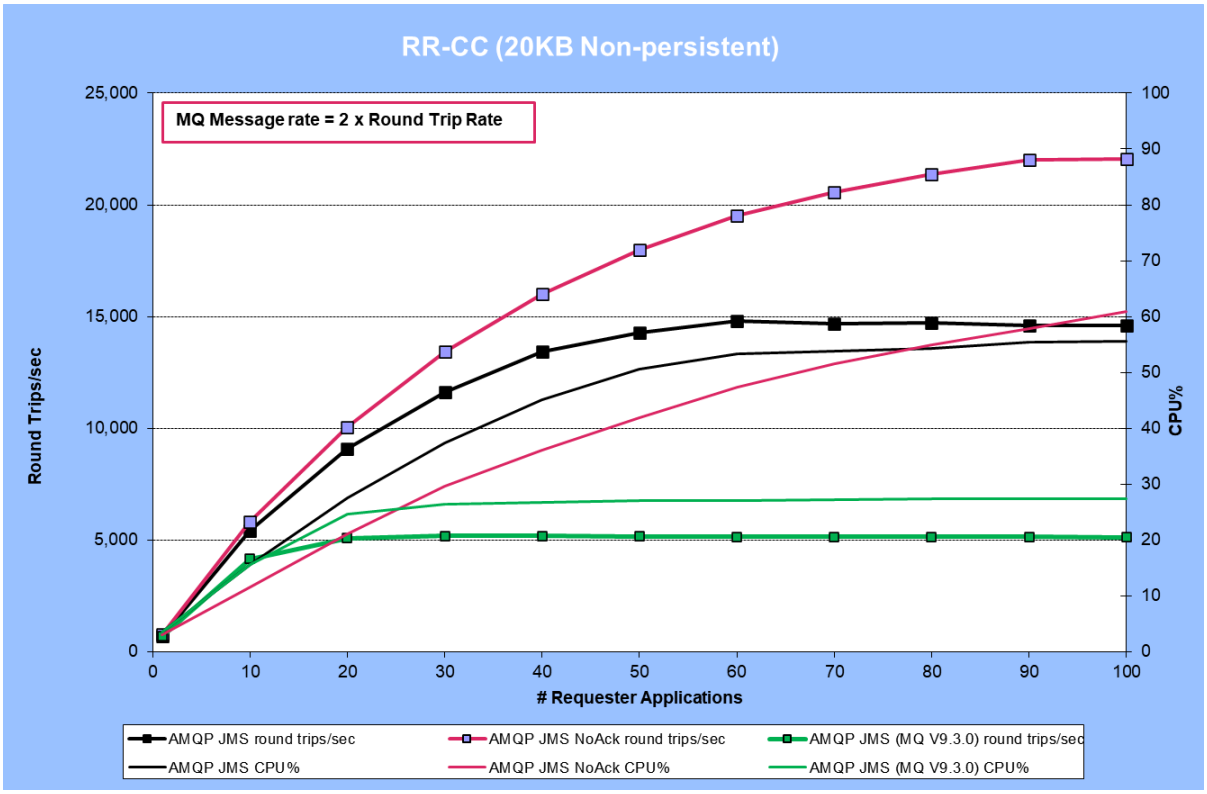


FIGURE 5 - RR-CC RESULTS (20KB NON-PERSISTENT, AMQP JMS VS AMQP JMS NOACK)

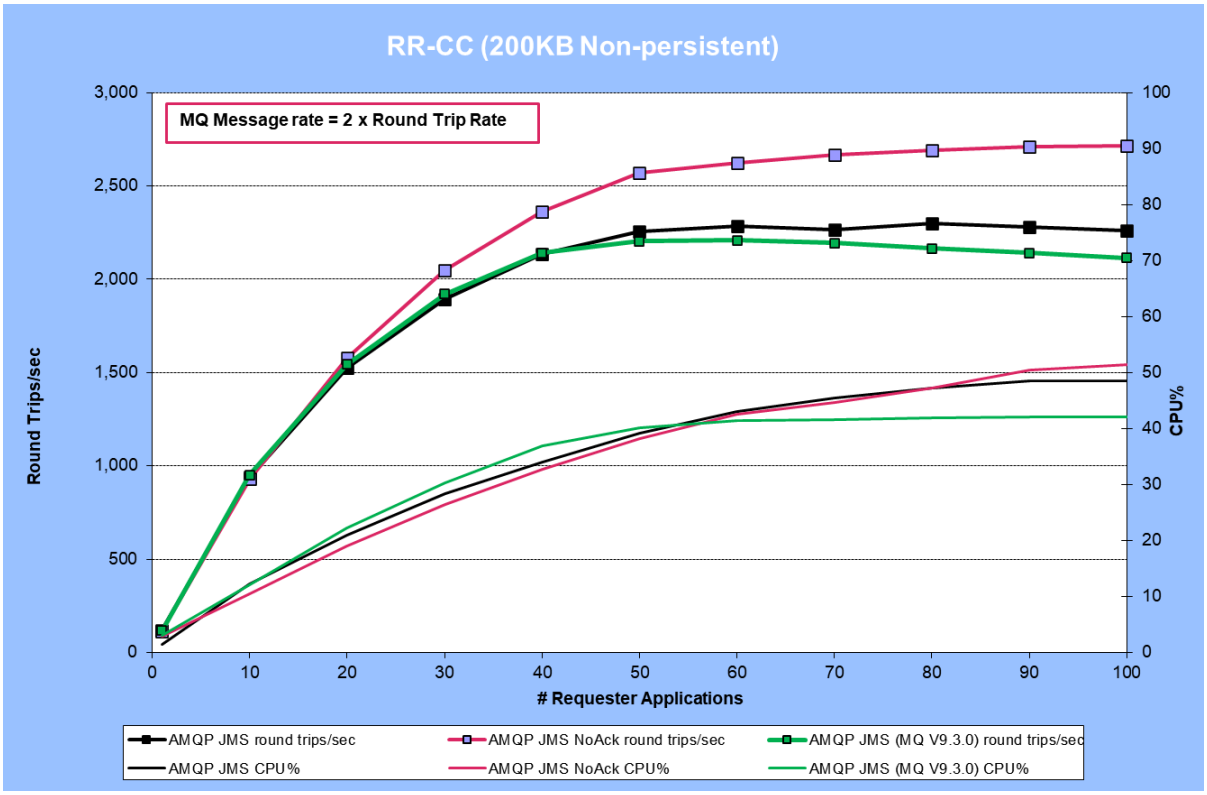


FIGURE 6 - RR-CC RESULTS (200KB NON-PERSISTENT, AMQP JMS VS AMQP JMS NOACK)

As the requestor/responder pattern is restricted to a roundtrip of each request/response message before the next one is sent by the same requestor application, it is heavily

affected by the latency of individual messages. This is why such a difference is seen between the default AMQP acknowledge mode and NoAck, with the NoAck not needing to wait for the acknowledgement of the message.

Using NoAck gave a significant performance boost for smaller message sizes, though for the larger message size (200K) the difference is less significant.

The 2K AMQP JMS test peaked at approximately 27,000 round trips/sec.

First, we can see that IBM MQ 9.3.3 significantly outperforms previous versions of MQ queue managers for most message sizes.

Secondly, with the NoAck setting the 2K message rate achieved was significantly higher than the Ack setting, but after peaking at around 40,000 round trips/sec the rate fell off as more clients were connected (though the NoAck mode still outperformed the default AMQP JMS acknowledge mode throughout). Internally AMQP listener worker thread handles receiving message from the queue manager and sending the response to the client and also receives and process the acknowledges from the client.

Initially graph goes up till 50 requestors. Going beyond that performance plateaus, as workers are having to handle concurrent sends and receives and each effectively serialises, switching between either sending or receiving. This can cause the rate to drop off but can be alleviated by reducing the number of AMQP worker threads. The “AMQP JMS NoAck (2)” plot in Figure 4 shows the result of setting 32 worker threads via the fix to APAR IT44457 (see section 10.3) The message rate increase is slightly less, but reducing the number of worker threads avoids the drop-off as more clients are added.

5.1.1 Setup for RR-CC tests.

- Workload type: RR-CC (see section 4.2).
- Client applications
  AMQP JMS : PerfHarness, utilising Qpid JMS
- Hardware: Server 1, Client 1, Client 2 (see section A.1).
- Additional Tuning: The JVM heap for the AMQP service was increased as follows:
  - Xmx2048m, Xms2048m

5.2 SR-CC (Sender/receiver) Non-persistent Workload

(Client mode senders on separate host. Client mode receivers on separate host).

Figure 7 and Figure 8 below illustrate the performance of non-persistent messaging with various numbers of sender applications in the point-to-point scenario (SR-CC) for 2KB and 20KB message sizes.

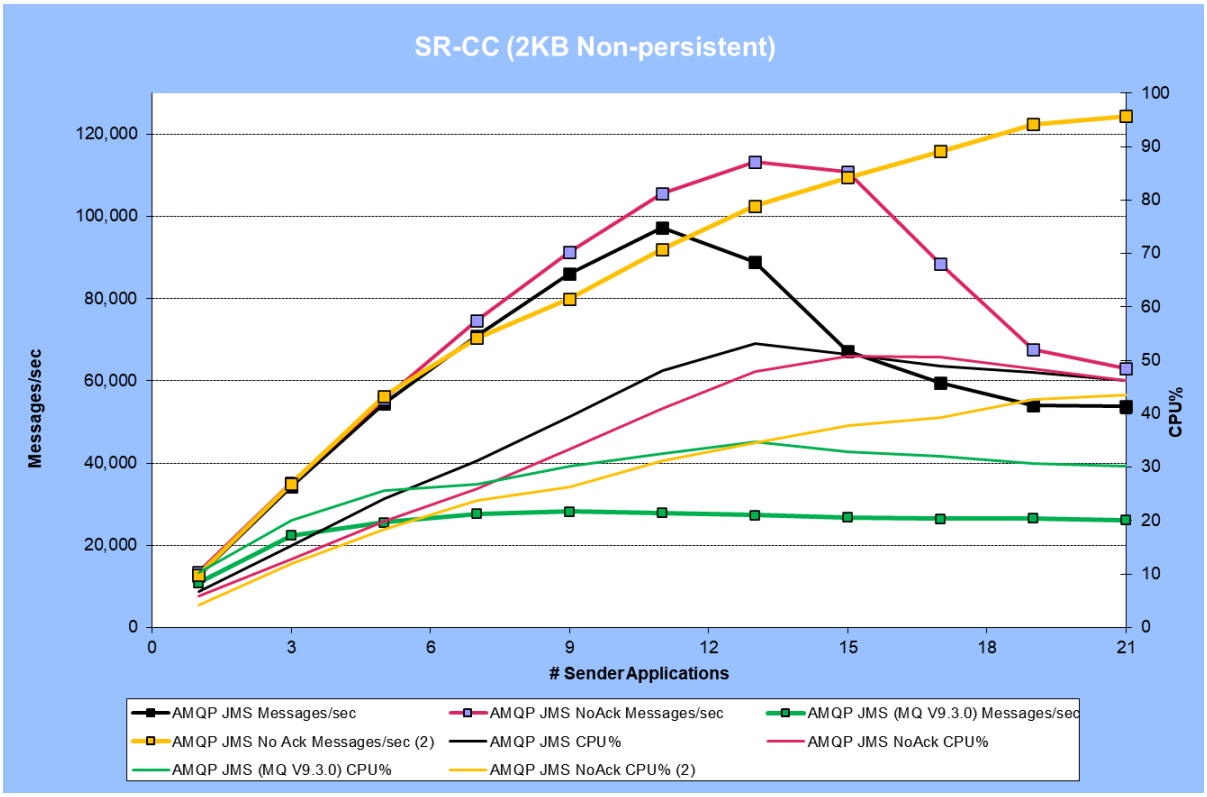


FIGURE 7 – SR-CC RESULTS (2KB NON-PERSISTENT, AMQP JMS VS AMQP JMS NOACK)

The "AMQP JMS NoAck (2)" plot in Figure 7 is an additional test with 32 AMQP worker threads set explicitly (see below).

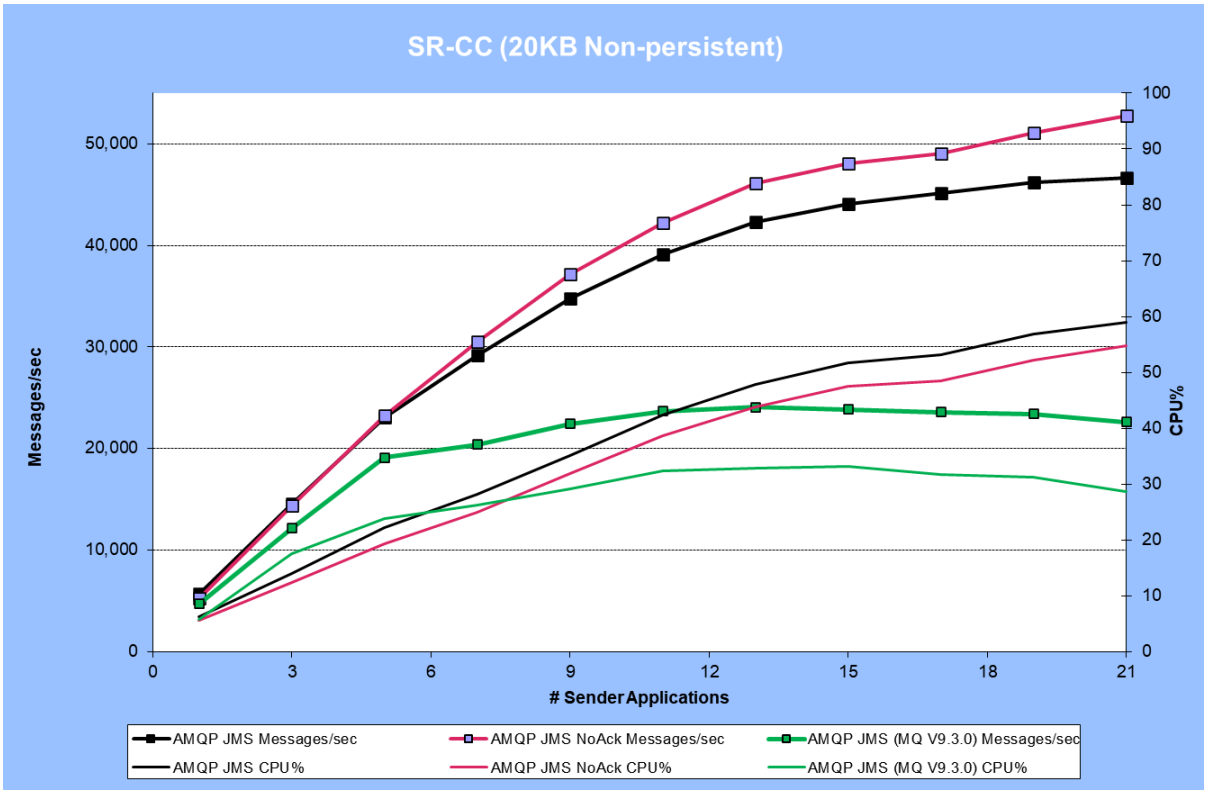


FIGURE 8 - SR-CC RESULTS (20KB NON-PERSISTENT, AMQP JMS VS AMQP JMS NOACK)

Again, IBM MQ 9.3.3 significantly outperforms previous versions. And this unregulated point-to-point test peaked at 113,306 messages/sec for AMQP JMS (2KB message with NoAck) . Note that this is the rate that messages were produced and consumed, with no build-up on the queues. The message rate is one-way (i.e. not directly comparable with the previous request-reply messaging tests where a round trip comprises of 2 MQPUTs and 2 MQGETs).

There is a limit for AMQP JMS evident in the plot above, where the CPU consumption is at 47.97%. To scale beyond this rate it would be necessary to spread the load across 2 queue managers.

After reaching the peak, it slightly dips due to internal workers handling concurrent sends and receives and each effectively serialises, switching between either sending or receiving. This can cause the rate to fall off but can be alleviated by reducing the number of AMQP worker threads. The "AMQP JMS NoAck (2)" plot in Figure 7 shows the result of setting 32 worker threads via the fix to APAR IT44457 (see section 10.3)

5.2.1 Setup for SR-CC Tests

- Workload type: SR-CC (see section 4.3).
- Client applications
  AMQP JMS: PerfHarness, utilising Qpid JMS
- Hardware: Server 1, Client 1, Client 2 (see section A.1).

5.3 PS-CC (Publish/subscribe) Non-persistent Workload

(Client mode publishers on separate host. - Client mode subscribers on separate host).

For the publish/subscribe scenario, a single publisher was started, publishing messages to one topic. The test started with a single subscriber and then was scaled up adding additional subscribers to increase the overall message rate.

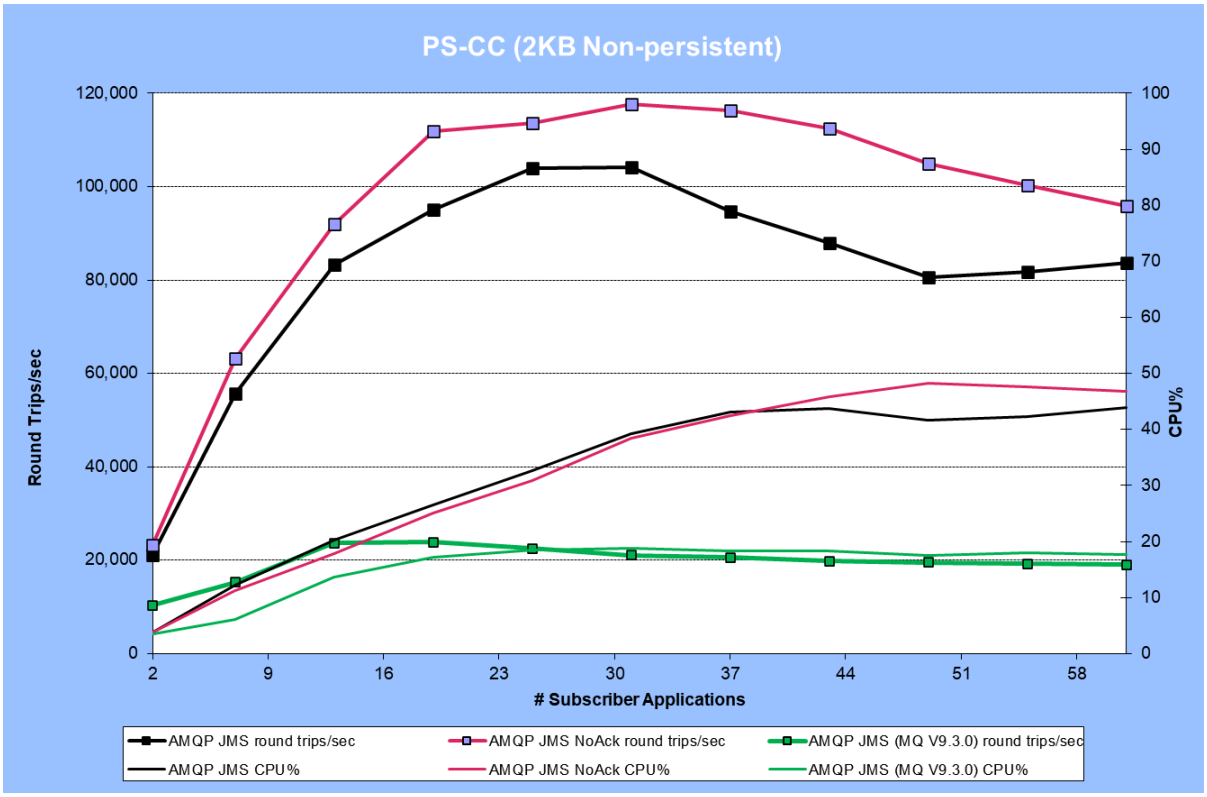


FIGURE 9 - PS-CC RESULTS (2KB NON-PERSISTENT, AMQP JMS VS AMQP JMS NOACK)

Once again, use of NoAck on the AMQP JMS client can increase performance, with higher rates achieved, and better scaling across more subscribers. As the rate flattens off this results in a decrease of the message rate per subscriber, but the chart shows results for an unregulated publisher (i.e. it sends messages as fast as it can).

In a production environment a publisher would typically be sending at a rate that would enable better scaling. E.g. in the scenario above, if a publisher was sending 1000 messages a second, the queue manager would be able to sustain that rate per subscriber up about 80 subscribers (for AMQP JMS with NoAck).

Again, these results show significant performance gains with IBM MQ 9.3.3 over previous versions of the queue manager.

5.3.1 Setup for PS-CC Test

- Workload type: PS-CC (see section 4.4).
- Client applications
  AMQP JMS: PerfHarness, utilising Qpid JMS
- Hardware: Server 1, Client 1, Client 2 (see section A.1).

6 Persistent Messaging Performance Test Results

The performance of persistent messaging is heavily influenced by the capabilities of the underlying filesystem hosting the queue files, and more critically, the MQ recovery log files. The general MQ performance reports have more details of this and comparisons of different filesystems as illustrations. E.g.

- IBM MQ V9.3 for Linux (x86-64 platform) Performance Report
- Persistent Messaging Performance in IBM MQ for Linux

The results presented here all used local NVMIe SSD storage for the recovery logs.

IBM MQ does not support transactions for AMQP JMS, which means that persistent messages are put onto queues outside of syncpoint, from the client perspective. Historically this has not been the optimal practice with regards to performance, but with the advent of implicit transactions introduced in MQ V9.0.5, persistent messaging outside of syncpoint should perform much better (see Syncpoints in IBM MQ for Multiplatforms).

Figure 10, Figure 11 and Figure 12 below illustrate the performance of persistent messaging with various numbers of requester applications for 2KB, 20KB and 200KB message sizes.

A round-trip in a requester/responder (RR-CC) scenario, is composed of 2 MQPUTs and 2 MQGETs, so the underlying MQ message rate is two times the round-trip rate.

AMQP acknowledgment

The following tests only show results for AMQP with acknowledgements enabled (the default). Whilst this is not the most highly performing mode (see non-persistent results above), for persistent messages it is expected that an application will require the higher assurances that consumer acknowledgments provide.

6.1 RR-CC (Requester/responder) Persistent Workload

(Client mode requesters on separate host. Client mode responders on separate host).

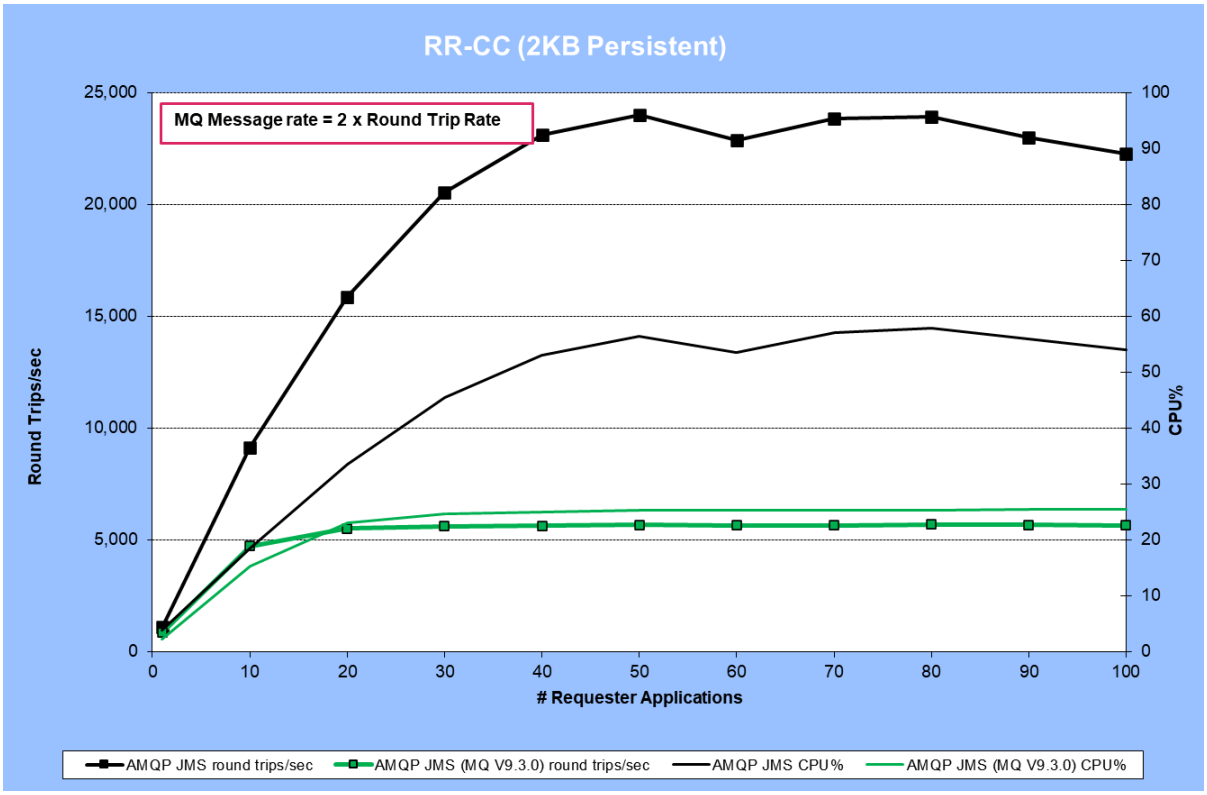


FIGURE 10 - RR-CC RESULTS (2KB PERSISTENT, AMQP JMS)

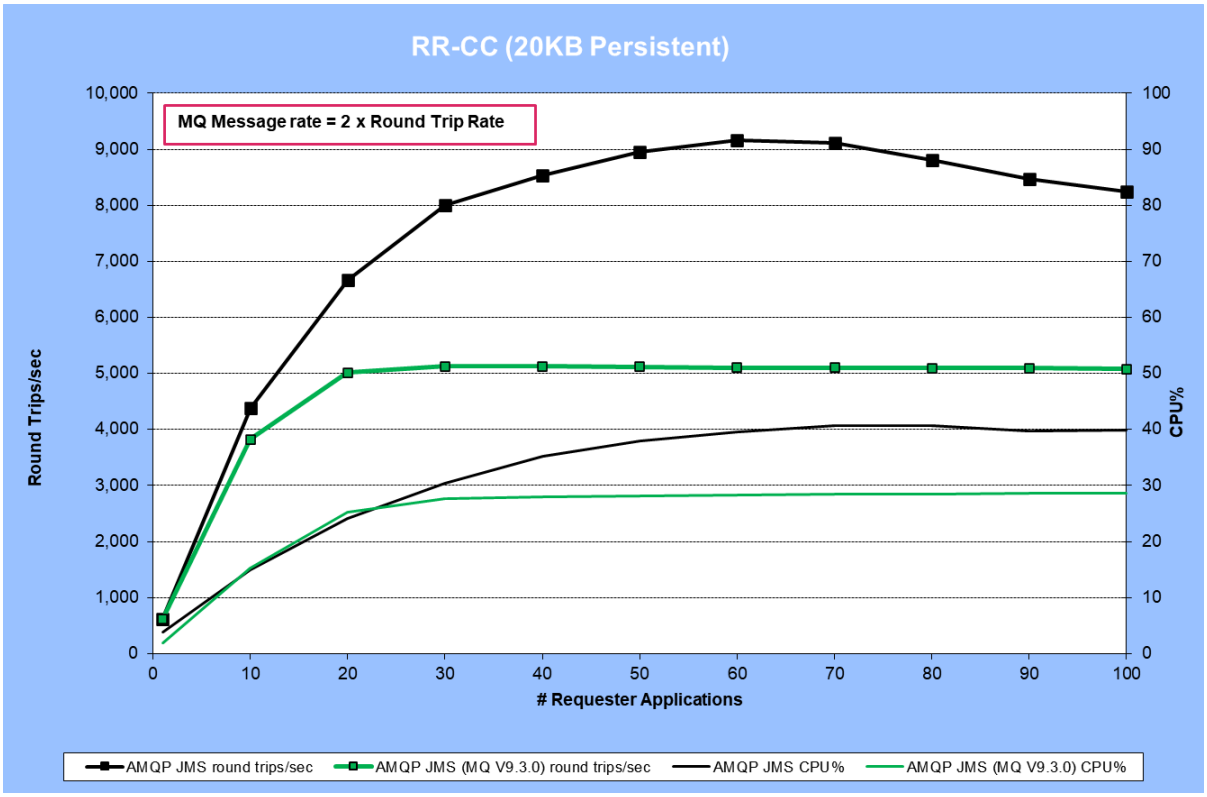


FIGURE 11- RR-CC RESULTS (20KB PERSISTENT, AMQP JMS)

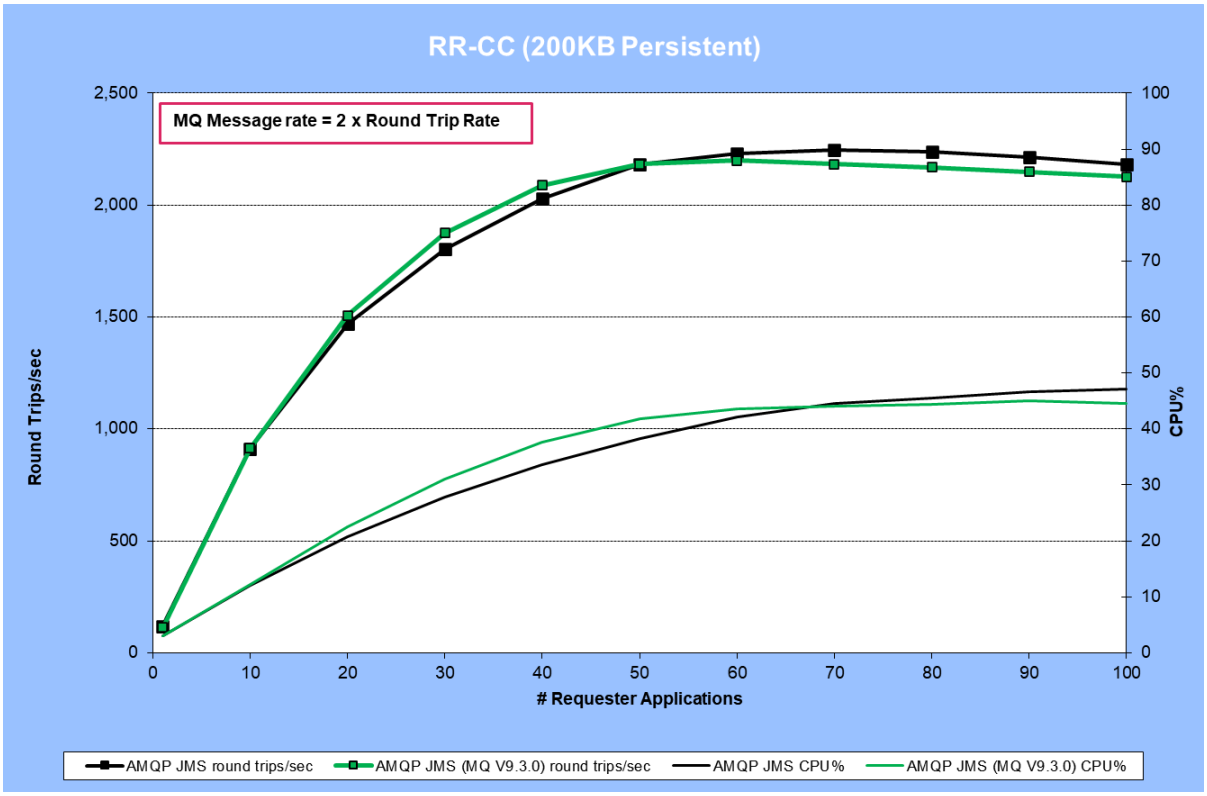


FIGURE 12 - RR-CC RESULTS (200KB PERSISTENT, AMQP JMS VS AMQP JMS NOACK)

As for the non-persistent tests, IBM MQ 9.3.3 significantly outperforms previous versions of queue managers for all but the largest message sizes tested.

Utilising NoAck with AMQP would achieve better throughput, although as described above, NoAck is less likely to be used with persistent messages, and therefore, not shown in the results above.

6.1.1 Setup for RR-CC Tests

- Workload type: RR-CC (see section 4.2).
- Client applications
  AMQP JMS: PerfHarness, utilising Qpid JMS
- Hardware: Server 1, Client 1, Client 2 (see section A.1).
- Additional Tuning: The JVM heap for the AMQP service was increased as follows:
  - Xmx2048m, Xms2048m

6.2 SR-CC (Sender/receiver) Persistent Workload

(Client mode senders on separate host. Client mode receivers on separate host).

Figure 13 and Figure 14 below illustrate the performance of persistent messaging with various numbers of sender applications in the point-to-point scenario (SR-CC) for 2KB and 20KB message sizes.

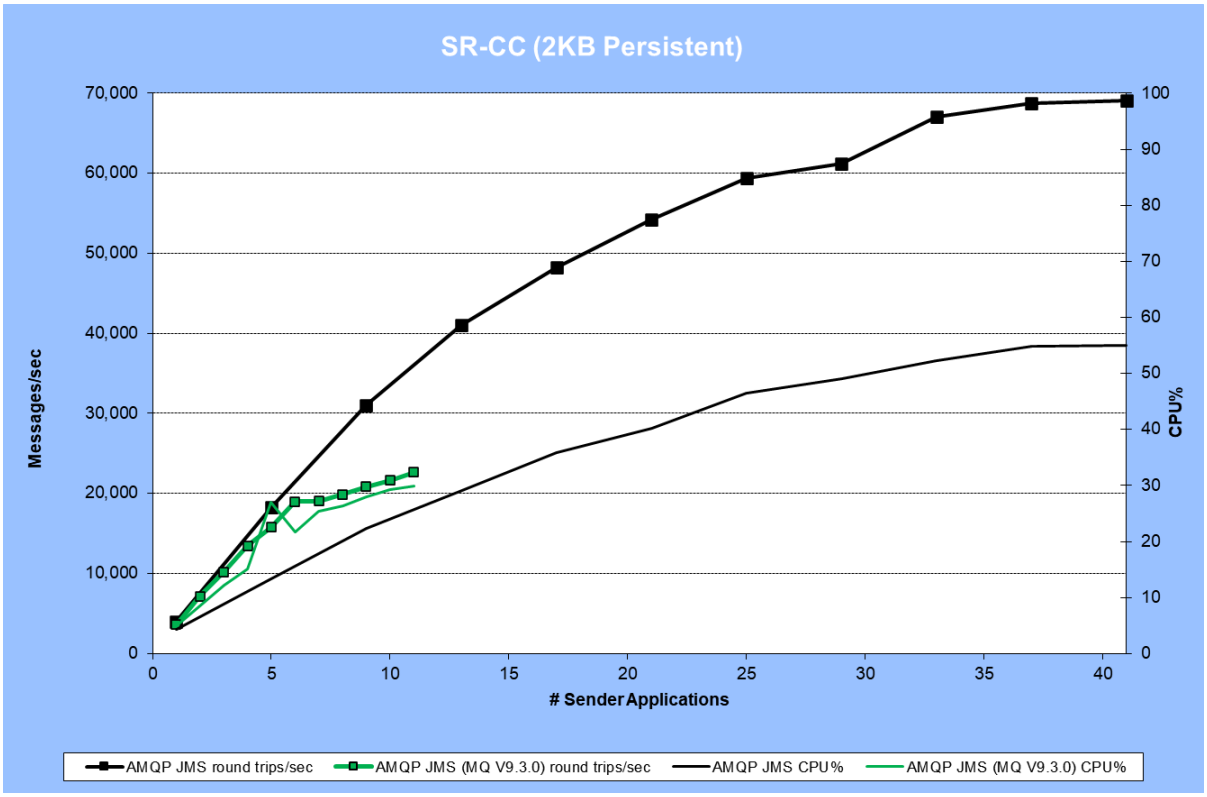


FIGURE 13 – SR-CC RESULTS (2KB PERSISTENT, AMQP JMS)

With Sender/Receiver tests it can be possible to fill the queues being used if the receivers do not keep up with the senders, as the senders are set to PUT messages as fast as they can. The tests are run with 2 or more receivers per queue in an attempt to keep the queues shallow (where each queue has a single Sender putting messages on it). Performance improvements in the AMQP service in MQ V9.3.3 meant that these persistent tests were able to be scaled up without the queues growing (i.e. the receivers kept up with the senders). In V9.3.0 the queue started filling up when more than 11 senders were started, which is why the V9.3.0 results are only plotted over that range. Understand that we are testing the limits of the product here though, even V9.3.0 would support a lot more senders running at lower message rates, without causing queues to fill (with the right number of performant receivers running). Testing performance for production scenarios should be defined by your own requirements.

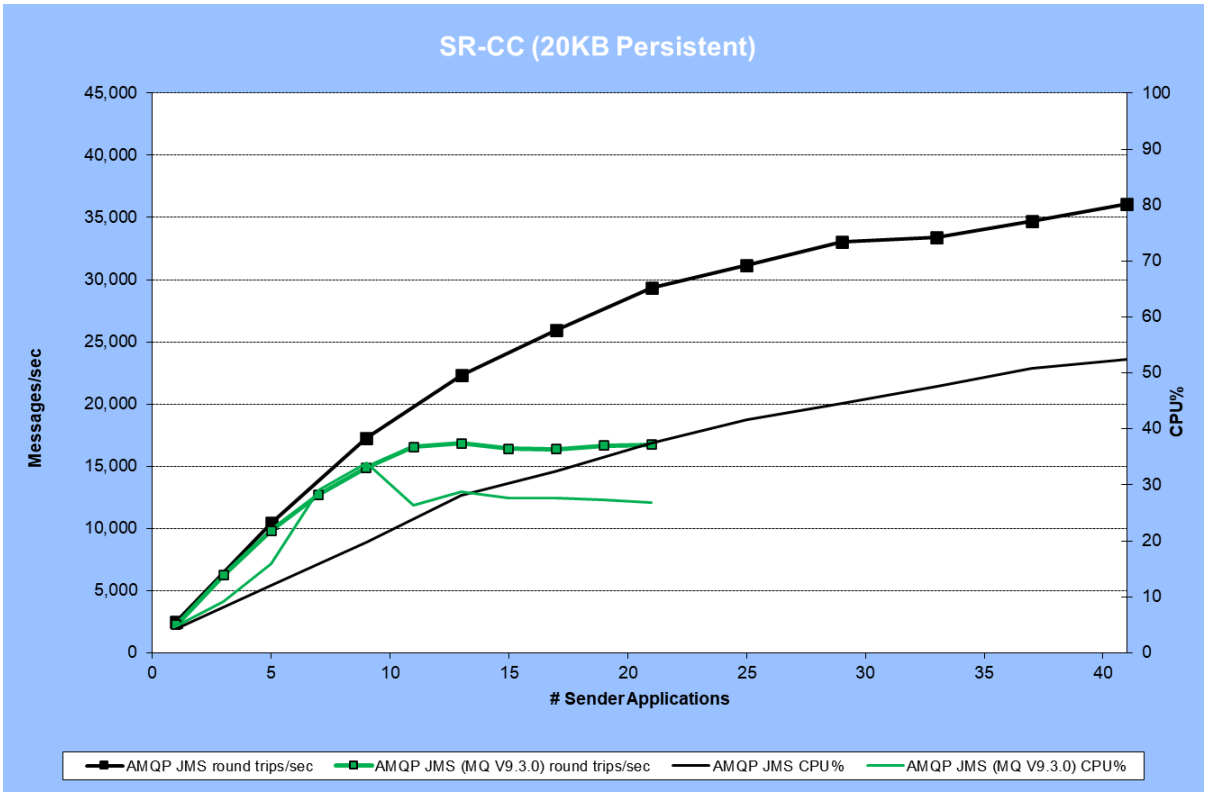


FIGURE 14 - SR-CC RESULTS (20KB PERSISTENT, AMQP JMS)

As with the previous chart for 2KB messages, the 20KB Sender/Receiver persistent test did not scale as far as the V9.3.3 test before the queues started to fill up. In this case it was only possible to scale to 21 senders for V9.3.0

6.2.1 Setup for SR-CC Tests

- Workload type: SR-CC (see section 4.3).
- Client applications
  AMQP JMS: PerfHarness, utilising Qpid JMS
- Hardware: Server 1, Client 1, Client 2 (see section A.1).

7 Non-Persistent Messaging Performance Test Results (AMQP JMS vs IBM MQ JMS)

The results in this section compare AMQP JMS (with no client acknowledgment – 'NoAck') to IBM MQ JMS for non-persistent messaging.

The test results are presented by workload with an illustrative plot (for 2KB) in each section followed by the peak throughput achieved for additional message sizes.

A round-trip in a requester/responder (RR-CC) scenario, is composed of 2 MQPUTs and 2 MQGETs, so the underlying MQ message rate is two times the round-trip rate.

As you will see, applications using the IBM MQ protocol outperform those using the AMQP protocol. However, multiple factors will influence your choice of protocol and client, see section 9 for more details.

7.1 RR-CC (Requester/responder) Non-persistent Workload

(Client mode requesters on separate host. Client mode responders on separate host).

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

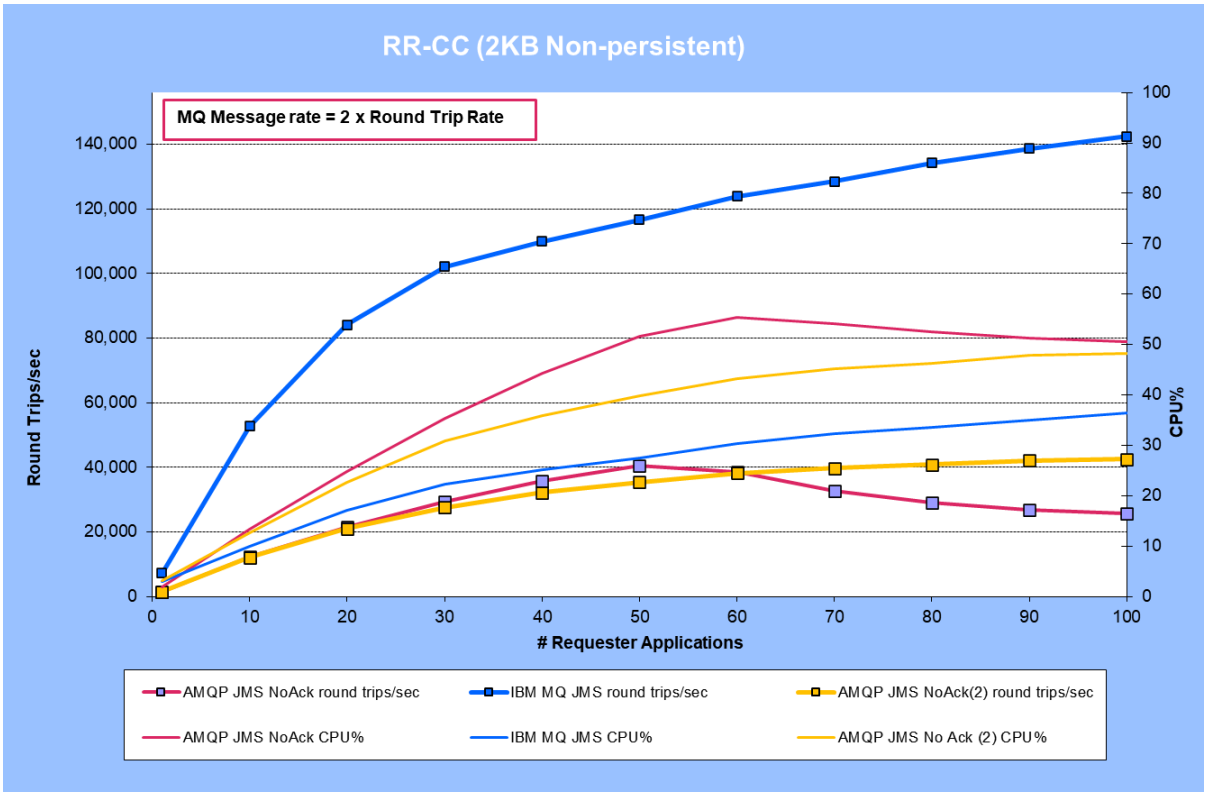


FIGURE 15 - RR-CC RESULTS (2KB NON-PERSISTENT, AMQP JMS NOACK VS IBM MQ JMS)

The “AMQP JMS NoAck (2)” plot in Figure 15 is an additional test with 32 AMQP worker threads set explicitly (see section 10.3).

The AMQP JMS with NoAck test peaked at approximately 40,000 round trips/sec.

Peak round trip rates for all message sizes tested can be seen in the table below. Note that as for the 2KB case plotted above, the IBM MQ JMS peak rate is the peak rate attained over the range of clients tested for AMQP JMS. The attainable peak rate for IBM MQ JMS will be higher than this as the rate was still climbing at the highest number of clients run.

The latency of IBM JMS and AMQP JMS vary as the protocol and design implementation are different. MQ JMS uses standard MQ channels for communicates using internal MQ protocol. AMQP JMS uses AMQP channel to communicate using AMQP frames. Frames are the basic units of communications in the AMQP protocol. There is some additional overhead AMQP service must perform for frame interpretation and convert the data back in the form of frames for sending response back to client.

Test	AMQP JMS NoAck			IBM MQ JMS		
	Max Rate*	CPU%	Clients	Max Rate*	CPU%	Clients
RR-CC (2KB Non-persistent)	40,480	51.65	#N/A	142,495	36.48	100
RR-CC (20KB Non-persistent)	22,064	60.88	#N/A	102,521	83.8	100
RR-CC (200KB Non-persistent)	2,715	51.35	#N/A	16,321	66.71	100

Round trips/sec

TABLE 2 - PEAK RATES FOR WORKLOAD RR-CC (NON-PERSISTENT)

7.1.1 Setup for RR-CC Test

- Workload type: RR-CC (see section 4.2).
- Client applications
  AMQP JMS : PerfHarness, utilising Qpid JMS
  IBM MQ JMS: PerfHarness utilising the standard IBM MQ classes for JMS.
- Hardware: Server 1, Client 1, Client 2 (see section A.1).
- Additional Tuning: The JVM heap for the AMQP service was increased as follows:
  - Xmx2048m, Xms2048m

7.2 SR-CC (Sender/receiver) Non-persistent Workload

(Client mode senders on separate host. Client mode receivers on separate host).

Figure 7 below illustrates the performance of non-persistent messaging with various numbers of sender applications in the point-to-point scenario (SR-CC) for the 2KB message size.

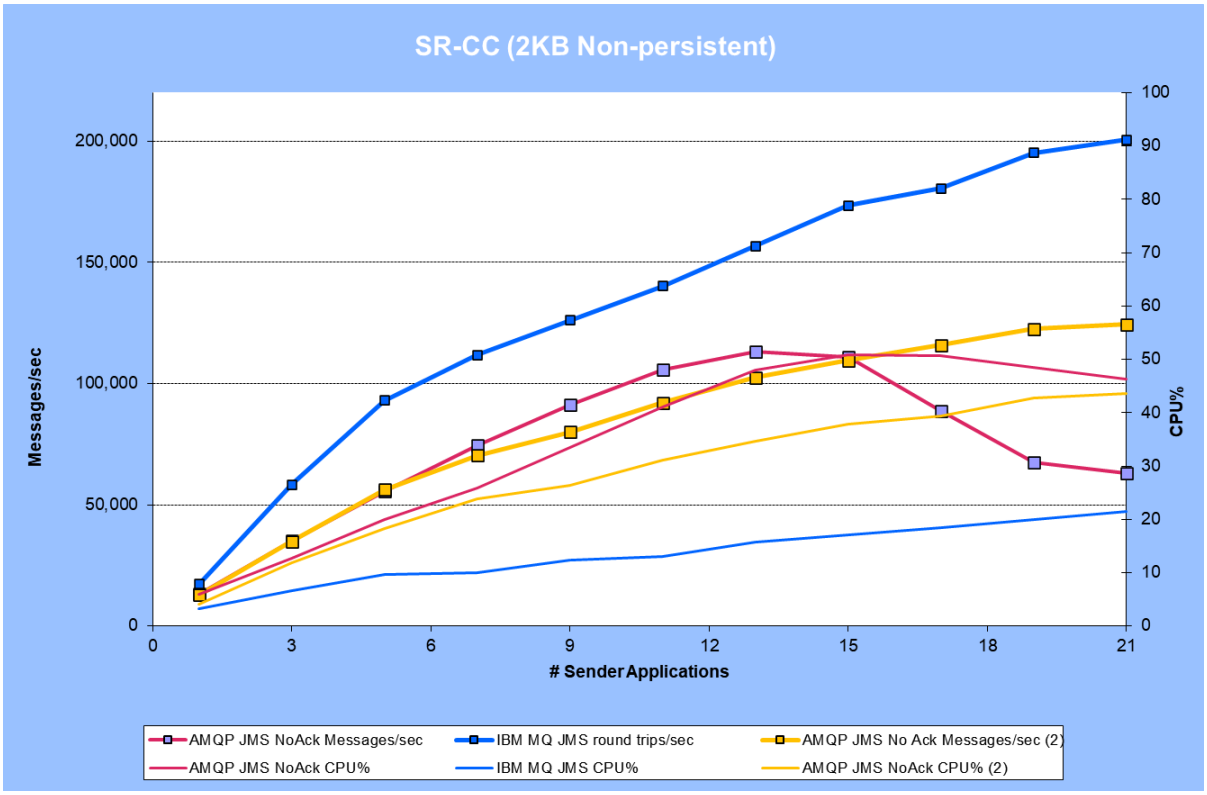


FIGURE 16 - SR-CC RESULTS (2KB NON-PERSISTENT, AMQPJMS NOACK VS IBM MQ JMS)

The "AMQP JMS NoAck (2)" plot in Figure 16is an additional test with 32 AMQP worker threads set explicitly (see section 10.3).

This unregulated point-to-point test peaked at 113,306 messages/sec for AMQP JMS with NoAck. Note that this is the rate that messages were produced and consumed, with no build-up on the queues. The message rate is one-way (i.e. not directly comparable with the previous request-reply messaging tests where a round trip comprises of 2 PUTs and 2 GETs).

There is a limit for AMQP JMS evident in the plot above, where the CPU consumption is at 48%. To scale beyond this rate it would be necessary to spread the load across 2 queue managers.

Peak message rates for all message sizes tested can be seen in the table below. Note that as for the 2KB case plotted above, the IBM MQ JMS peak rate is the peak rate attained over

the range of clients tested for AMQP JMS. The attainable peak rate for IBM MQ JMS will be higher than this as the rate was still climbing at the highest number of clients run.

Test	AMQP JMS NoAck			IBM MQ JMS		
	Max Rate*	CPU%	Clients	Max Rate*	CPU%	Clients
SR-CC (2KB Non-persistent)	113,307	47.97	13	200,482	21.45	21
SR-CC (20KB Non-persistent)	52,819	54.82	21	163,087	59.79	21

Messages/sec

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

7.2.1 Setup for

- Workload type: SR-CC (see section 4.3).
- Client applications
  AMQP JMS : PerfHarness, utilising Qpid JMS
  IBM MQ JMS: PerfHarness utilising the standard IBM MQ classes for JMS.
- Hardware: Server 1, Client 1, Client 2 (see section A.1).

7.3 PS-CC (Publish/subscribe) Non-persistent Workload

(Client mode publishers on separate host. - Client mode subscribers on separate host).

For the publish/subscribe scenario, a single publisher was started, publishing messages to one topic. The test started with a single subscriber and then was scaled up adding additional subscribers to increase the overall message rate.

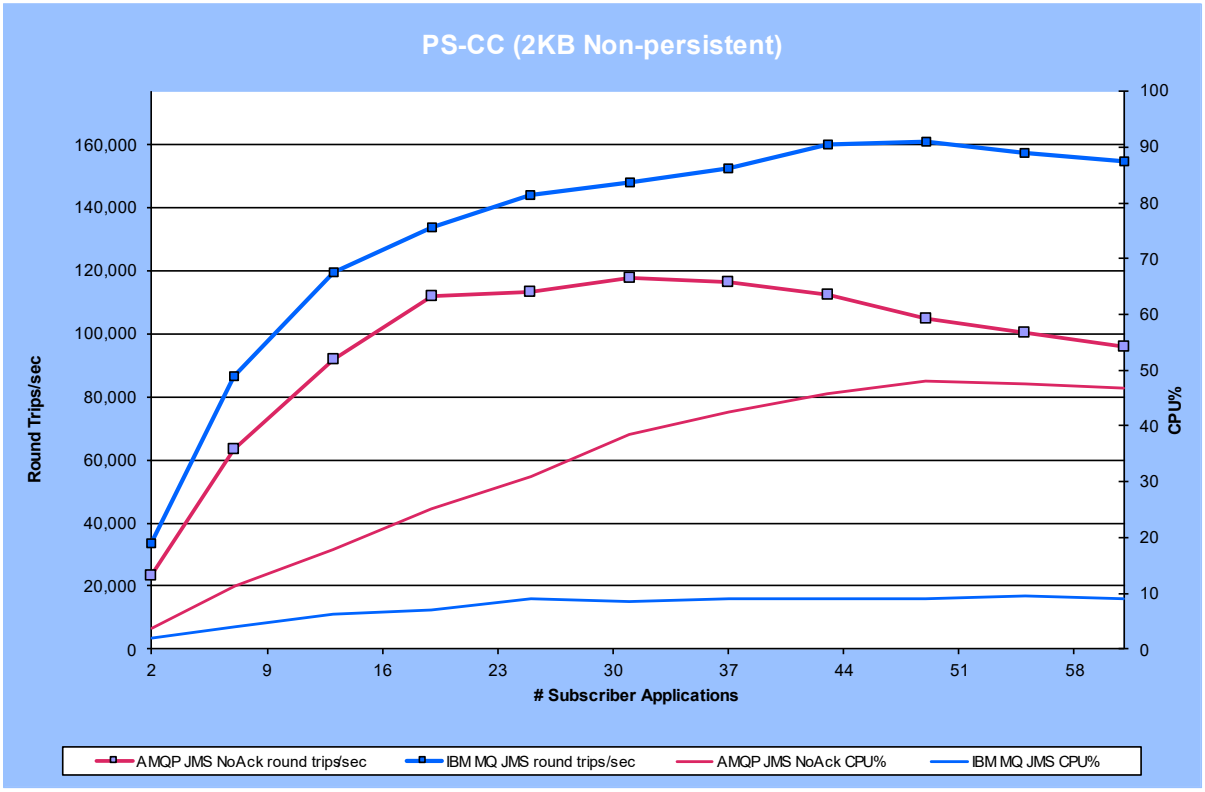


FIGURE 17 - PS-CC RESULTS (2KB NON-PERSISTENT, AMQP JMS NOACK VS IBM MQ JMS)

Though more expensive in terms of CPU, the AMQP JMS pub/sub rates come closer to IBM MQ JMS than the previous standard messaging test.

7.3.1 Setup for test PS-CC

- Workload type: PS-CC (see section 4.4).
- Client applications
  AMQP JMS: PerfHarness, utilising Qpid JMS
  IBM MQ JMS: PerfHarness utilising the standard IBM MQ classes for JMS.
- Hardware: Server 1, Client 1, Client 2 (see section A.1).

8 Persistent Messaging Performance Test Results (AMQP JMS vs IBM MQJMS)

The results in this section compare AMQP JMS (with client acknowledgment – 'Ack') to IBM MQ JMS for persistent messaging.

As you will see, applications using the IBM MQ protocol outperform those using the AMQP protocol. However, multiple factors will influence your choice of protocol and client, see section 9 for more details.

The performance of persistent messaging is heavily influenced by the capabilities of the underlying filesystem hosting the queue files, and more critically, the MQ recovery log files. The general MQ performance reports have more details of this and comparisons of different filesystems as illustrations. E.g.

- IBM MQ V9.3 for Linux (x86-64 platform) Performance Report
- Persistent Messaging Performance in IBM MQ for Linux

The results presented here all used local NVMIe SSD storage for the recovery logs.

IBM MQ does not support transactions for AMQP JMS, which means that persistent messages are put onto queues outside of syncpoint, from the client perspective. Historically this has not been the optimal practice with regards to performance, but with the advent of implicit transactions introduced in MQ V9.0.5, persistent messaging outside of syncpoint should perform much better. (See Syncpoints in IBM MQ for Multiplatforms).

The persistent IBM MQ JMS tests run as a comparison to AMQP JMS are also non-transactional to match, but tests showed transactional IBM MQ JMS message rates were no higher, due to the benefits of implicit transactions being leveraged.

The test results are presented by workload with an illustrative plot in each section followed by the peak throughput achieved for additional message sizes.

A round-trip in a requester/responder (RR-CC) scenario, is composed of 2 MQPUTs and 2 MQGETs, so the underlying MQ message rate is two times the round-trip rate.

8.1 RR-CC (Requester/responder) Persistent Workload

(Client mode requesters on separate host. Client mode responders on separate host).

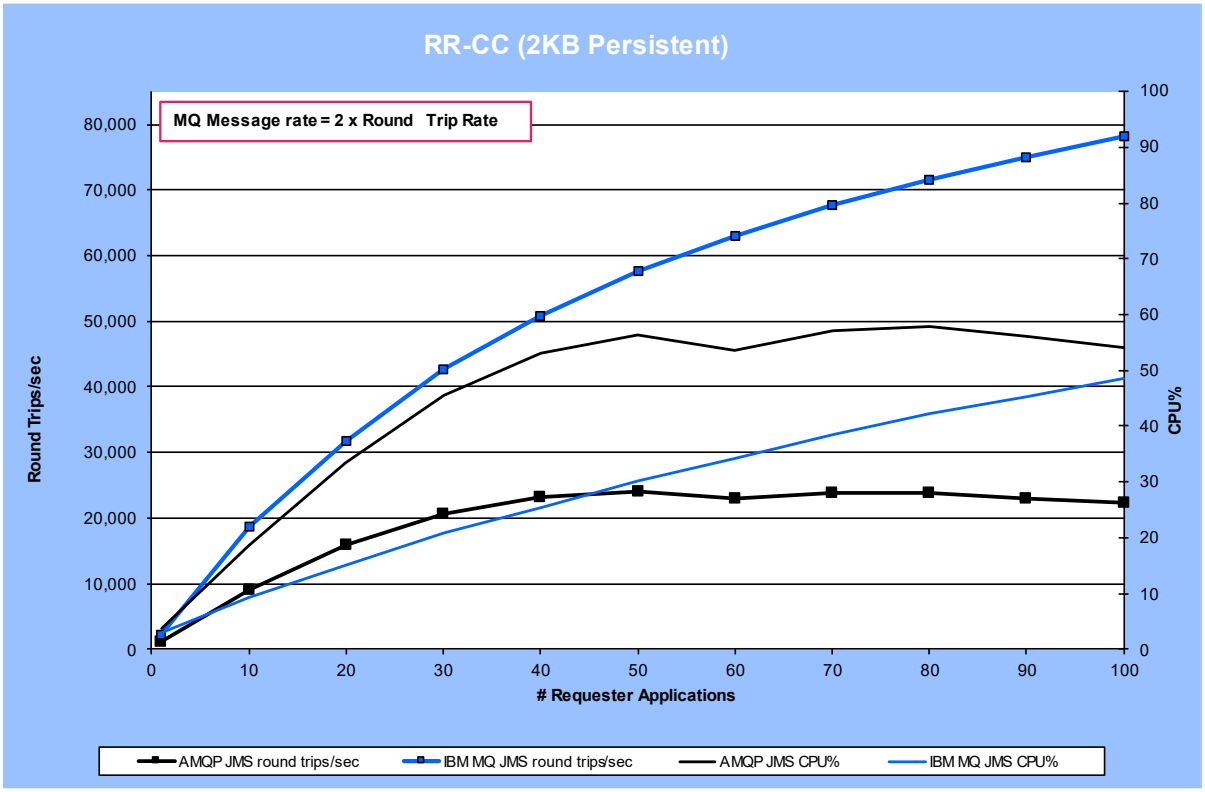


FIGURE 18 - RR-CC RESULTS (2KB PERSISTENT, AMQP JMS VS IBM MQ JMS)

Whilst AMQP JMS is significantly slower than JMS, the difference is less than for non-persistent messaging as the synchronous recovery log writes required for persistent messaging has a significant impact on performance as well.

Peak round trip rates for all message sizes tested can be seen in the table below. Note that as for the 2KB case plotted above, the IBM MQ JMS peak rate is the peak rate attained over the range of clients tested for AMQP. The attainable peak rate for IBM MQ JMS will be higher than this as the rate was still climbing at the highest number of clients run.

Test	AMQP JMS			IBM MQ JMS		
	Max Rate*	CPU%	Clients	Max Rate*	CPU%	Clients
RR-CC (2KB Persistent)	24,004	56.35	50	78,140	48.55	100
RR-CC (20KB Persistent)	9,166	39.61	60	56,691	59.98	100
RR-CC (200KB Persistent)	2,245	44.52	70	12,373	55.68	100

***ROUND TRIPS/ SEC**

TABLE 4 - PEAK RATES FOR WORKLOAD RR-CC (PERSISTENT)

8.1.1 Setup for Test RR-CC

- Workload type: RR-CC (see section 4.2).

- Client applications
  AMQP JMS : PerfHarness, utilising Qpid JMS
  IBM MQ JMS: PerfHarness utilising the standard IBM MQ classes for JMS.
- Hardware: Server 1, Client 1, Client 2 (see section A.1).
- Additional Tuning: The JVM heap for the AMQP service was increased as follows:
    - Xmx2048m, Xms2048m

8.2 SR-CC (Sender/receiver) Persistent Workload

(Client mode senders on separate host. Client mode receivers on separate host).

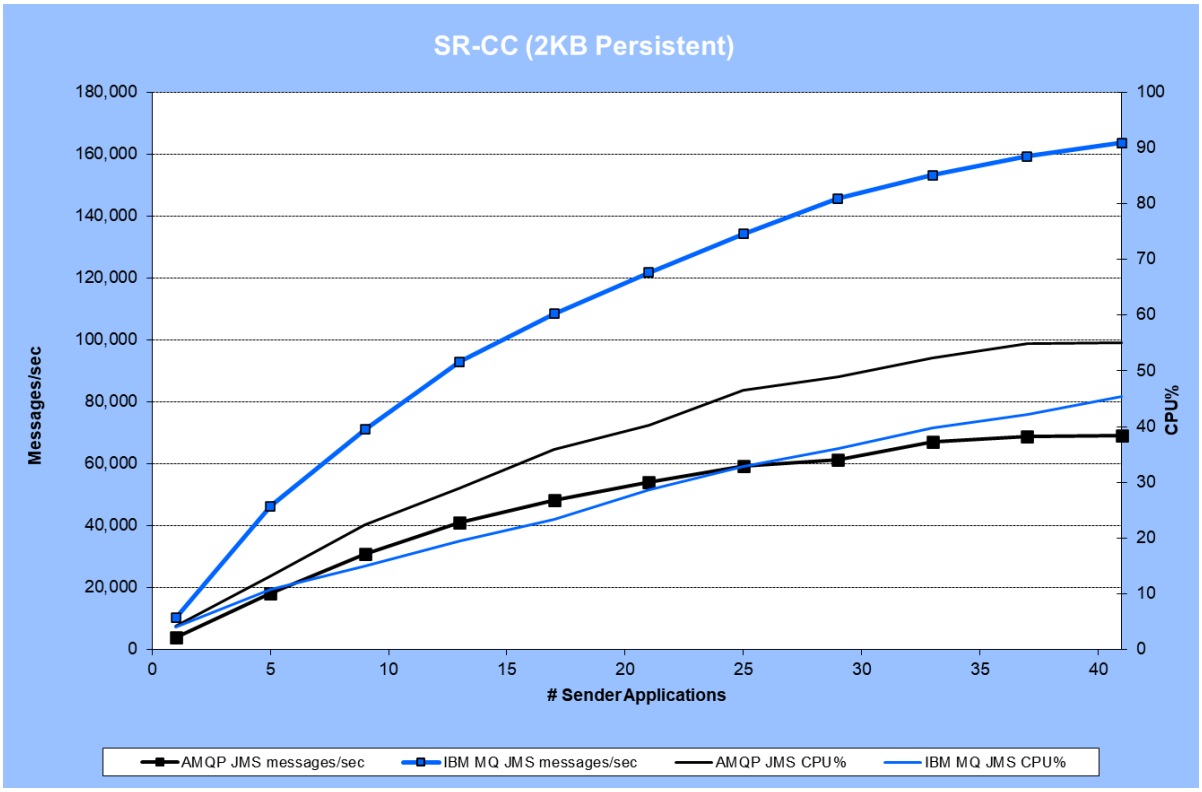


FIGURE 19 – SR-CC RESULTS (2KB PERSISTENT AMQP JMS VS IBM MQ JMS)

As for the requester/responder workload, the AMQP JMS rates are significantly lower than IBM MQ JMS but closer than the no-persistent rates for the same test, due again to the impact of synchronous recovery log writing required to support persistence.

Peak round trip rates for 2KB & 20KB messages can be seen in the table below. Note that as for the 2KB case plotted above, the IBM MQ JMS peak rate is the peak rate attained over the range of clients tested for AMQP JMS. The attainable peak rate for IBM MQ JMS will be higher than this as the rate was still climbing at the highest number of clients run.

Test	AMQP JMS			IBM MQ JMS		
	Max Rate*	CPU%	Clients	Max Rate*	CPU%	Clients
SR-CC (2KB Persistent)	69,122	55.02	41	163,666	43.62	41
SR-CC (20KB Persistent)	36,110	52.4	41	113,125	55.76	37

TABLE 5 - PEAK RATES FOR WORKLOAD SR-CC (PERSISTENT)

8.2.1 Setup for test SR-CC

- Workload type: SR-CC (see section 4.3).
- Client applications
  AMQP JMS: PerfHarness, utilising Qpid JMS
  IBM MQ JMS: PerfHarness utilising the standard IBM MQ classes for JMS.
- Hardware: Server 1, Client 1, Client 2 (see section A.1).

9 Making the right decision between AMQP protocol(IBM AMQP service) and IBM MQ protocol(IBM MQ JMS):

The performance results in sections 7 and 8 clearly show that JMS over the IBM MQ protocol outperforms the open standard, AMQP protocol. But many factors should be taken into account when choosing the right protocol for your applications.

For selecting and deciding between AMQP protocol(used by IBM AMQP Service) and MQ protocol (used by IBM MQ JMS), few points are listed here :-

- AMQP is open source messaging protocol and enabling various open-source clients libraries offering different choices of programming language. IBM MQ provides AMQP support using AMQP channel and recommends Qpid JMS client for JMS like messaging requirements over AMQP. Whereas MQ JMS uses standard MQ channels for messaging in IBM MQ and communicates using internal MQ protocol.
- Any AMQP 1.0 compliant client can be used to develop messaging applications and integrate using MQ AMQP service. Usages of open source AMQP 1.0 compliant clients will be supported by open-source community and will have open-source licensing terms and conditions whereas MQ clients has its own support provided by IBM MQ and IBM MQ license terms & conditions.

When deciding between MQ protocol (IBM MQ JMS) or AMQP protocol (AMQP JMS service), your choice should be influenced by factors such as message durability and reliability, required feature or specific requirement, support options, existing infrastructure, programming language preferences etc. Few points are listed here that will help in deciding which one to choose:-

- If you prioritize high durability and guaranteed message delivery and only once message delivery, MQ is well-suited for this purpose. It offers features like transaction management and reliable message persistence, ensuring that messages are stored securely and can be retrieved even in the event of system failures.
  MQ JMS also provides support for non-persistent which can be used in case of high performance and immediate processing are crucial.
- On the other hand, if your main concern is flexibility and interoperability between different systems and languages, you might lean towards using AMQP. AMQP can also provide message persistence and acknowledgments with at least once delivery. For scenarios of high performance and immediate processing are crucial. Non-persistent with Noack is better fit. AMQP primary advantage lies in its ability to connect diverse messaging systems, making it a great choice when your environment requires communication between diverse components.

10 AMQP JMS Specific Tuning

10.1 AMQP JMS JVM Heap Settings

The only AMQP JMS specific tuning used to collect the data for this report was increasing the JVM heap size for the AMQP service.

The default heap size conserves memory but for intensive messaging scenarios (by definition, all of those in this report) performance will be helped by increasing the heap size. All test in this report used a 2GB heap.

When running at very high messaging rates with large messages you may even encounter the following client-side error, indicating that the heap will need to be increased to run at the rate attempted:

```
org.apache.qpid.jms.exceptions.JmsConnectionFailedException: The JMS
connection has failed: AMQXR1016W:
com.ibm.mq.MQXRService.MQXRService$1@9b96a60c(Object) discovered that most
of the available memory has been used. [condition = amqp:connection:forced]
```

If this happens you should cater for the larger amount of data being processed by the AMQP service by increasing the heap size of the JVM configured in amqp_java.properties. This file is in the amqp directory for the queue manager, e.g. for queue manager QM1 it will be in

```
/var/mqm/qmgrs/QM1/amqp_java.properties
```

10.2 AMQP JMS Batch Size and Interval

A significant performance improvement in AMQP support for MQ in V9.3.3 is due to refactoring of code, and batch processing message acknowledgment.

Batching is enabled by default but can be controlled by the following 2 tunables in amqp_java.properties file:

com.ibm.mq.AMQP.BATCHSZ :

This attribute defines the maximum number of acknowledgments to be received before the AMQP service removes messages. The number can be in the range 1 through 9999. If an invalid number is set, or if the specified number is out of range, the default value of 50 is used.

The batch size does not affect the way that the messages are transferred. Messages are always transferred individually but are then removed in a batch after the AMQP service receives the acknowledgments. The actual size of a batch can be less than the value specified by com.ibm.mq.AMQP.BATCHINT. For example, a batch completes if the period set by the com.ibm.mq.AMQP.BATCHINT attribute expires.

com.ibm.mq.AMQP.BATCHINT :

This attribute defines the amount of time, in milliseconds, for which the AMQP service keeps acknowledged messages on the queue. If the batch is not full, then the batch is cleared after this duration. You can specify any number of milliseconds, from 1 through 999 999 999. The default value is 50. If you do not specify a value for this attribute, the default value of 50 is used.

Increasing these values may increase the likelihood that a consumed message is re-delivered following a failure in the MQ server. Decreasing these values, may decrease performance.

From IBM MQ 9.3.3, when the queue manager is created, the amqp_java.properties file contains the following default values for the system properties:

```
-Dcom.ibm.mq.AMQP.BATCHSIZE=50
-Dcom.ibm.mq.AMQP.BATCHINT=50
```

10.3 AMQP Worker Threads

The number of AMQP worker threads can be explicitly set with the fix to APAR IT44457 (https://www.ibm.com/support/pages/apar/IT44457).

Internally AMQP listener worker threads handle receiving message from the queue manager and sending the response to the client and also receive and process the acknowledgments from the client, switching between these modes and larger numbers of threads can cause contention, reducing the overall throughput in some scenarios (particularly on hyperthreading enabled machines with a large number of cores. Smaller, non-persistent message sizes are more prone to this issue occurring, where this switching is proportionally a bigger part of the work. This can be alleviated by reducing the number of AMQP worker threads.

The machine used in these tests was a 32-core server with hyperthreading enabled, presenting 64 logical cores to the JVM. AMQP will create 64 worker threads by default, but this can be configured with the fix to APAR IT44457 by adding an explicit value to the amqp_unix.properties, e.g.:

```
com.ibm.mq.MQXR.Workers=32
```

Setting the number of threads explicitly to 32 improved the performance of two test cases in this report (see sections 5.1 & 5.2).

Appendix A: Test Configurations

A.1 Hardware/Software

All the testing in this document was performed on the following hardware and software configurations:

A.1.1 Hardware

Server1 machine:

- ThinkSystem SR630 V2 – [Model: 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 adapter connecting to client machines via 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.
- Operation system: Red Hat Enterprise Linux Server release 8.5 (Ootpa)

Client1 & client2 are identical machines:

- ThinkSystem SR630 – [Model: 7X02CTO1WW]
- 2 x 12 core CPUs.
  Core: Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz
- 192GB RAM
- 100Gb ethernet adapter connecting to server machine via 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.
- Operating System: Red Hat Enterprise Linux Server release 7.9 (Maipo)

A.1.2 Software

- JMSPerfHarness test driver (see Appendix B:)
- IBM MQ V9.3.3

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
```

A.2.2 IBM MQ

The following parameters are added or modified in the qm.ini files for the tests run in section 5 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: Resources

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

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

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

Persistent Messaging Performance in IBM MQ for Linux
https://ibm-messaging.github.io/mqperf/mqio_v1.pdf

APAR IT44457: The com.ibm.mq.MQXR.Workers property is not honored by the AMQP listener
https://www.ibm.com/support/pages/apar/IT44457