

IBM MQ Streaming Queues Performance Report.

Version 1.0 - September 2021

Paul Harris
IBM MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire
United Kingdom



1 Notices

Please take Note!

Before using this report, please be sure to read the paragraphs on “disclaimers”, “warranty and liability exclusion”, “errors and omissions”, and the other general information paragraphs in the "Notices" section below.

First Edition, September 2021.

© Copyright International Business Machines Corporation 2021. All rights reserved.

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 paper is intended for architects, systems programmers, analysts, and programmers wanting to understand the performance characteristics, of streaming queues, as introduced in IBM MQ V9.2.3. The information is not intended as the specification of any programming interface that is provided by IBM. It is assumed that the reader is familiar with the concepts and operation of IBM MQ.

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, IBM MQ

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.

2 Contents

1	NOTICES	2
2	CONTENTS.....	4
3	INTRODUCTION.....	5
4	TEST SCENARIOS	6
5	SCALING THE SOLUTIONS	10
6	RESULTS	11
6.1	Persistent 2KB Messaging with Single Duplicate Messages.....	11
6.2	Batching Duplicate Message Consumption and Overhead of MUSTDUP	12
6.3	Persistent 2KB Messaging with Multiple Duplicate Messages.	14
6.4	Peak Rates Achieved for All Scenarios.....	15
7	CONCLUSIONS	17
8	RESOURCES.....	18
APPENDIX A:	SOFTWARE AND HARDWARE	19
APPENDIX B:	ADDITIONAL DATA.....	20

3 Introduction

The streaming queues feature of IBM® MQ, introduced in V9.2.3 allows you to configure a queue to put a near-identical copy of every message to a second queue (see [New Streaming Queue feature for MQ 9.2.3](#))

This report will illustrate the cost of the additional work by the queue manager in duplicating messages. Setting STREAMQ to point to another queue for instance will typically double the internal message rate associated with the original queue (each PUT to the original queue will trigger an additional PUT to the queue defined by STREAMQ). If STREAMQ is set to a TOPIC alias, then the increase in message rate will depend on the number of subscribers to that topic.

The additional work will show as increased CPU and (for persistent messages, or where STRMQOS is set to MUSTDUP) additional I/O to the queue manager's recovery log.

It is important to consider how these additional messages are consumed to minimise the additional load on the system:

- To achieve the best performance, duplicate messages on the streaming queue should be consumed by applications interested in the copy at the same time, to avoid deep queues building up.
- Batching multiple duplicate messages into units of work can reduce the overhead further (see below).

Previous approaches to duplicating messaging involved using Pub/Sub (though this requires the consuming application to switch to the new subscriber queue). Here we will compare the cost of duplicating messages using streaming queues vs Pub/Sub.

4 Test Scenarios

Two scenarios (with variants) were measured:

1. Single duplicate message
 - a. Using streaming queues (Fig 1)
 - b. Using Pub/Sub (Fig 2)
2. 6 Duplicate messages.
 - a. Using streaming queues with Pub/Sub (Fig 3)
 - b. Using Pub/Sub only (Fig 4)

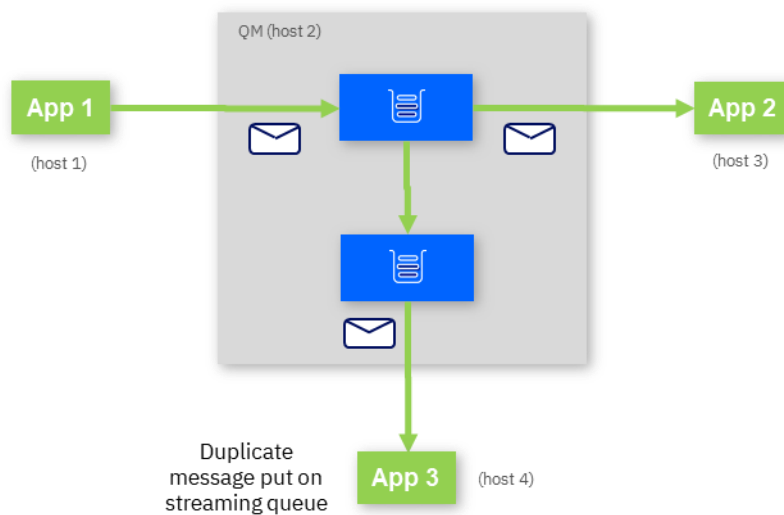


FIGURE 1 : SINGLE DUPLICATE MESSAGE PER QUEUE USING STREAMING QUEUE.

Figure 1 above shows a simple case of generating a single duplicate for each message by setting STREAMQ to point to a second queue where copies of messages are PUT. No change is needed to App1 or App2.

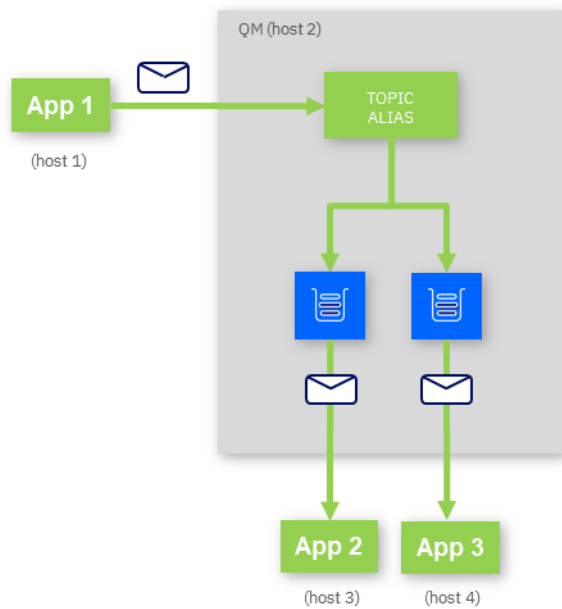


FIGURE 2 : SINGLE DUPLICATE MESSAGE USING PUB/SUB

Figure 2 shows the topology of using Pub/Sub to generate a single duplicate message. In the tests the first subscriber (App 2) is on host 3, whilst the subscriber for the duplicate message is on host 4. This pattern ensures a common topology where the primary application is on one host and secondary application, consuming the duplicate message (for logging or auditing, for example) is on a separate host. If this approach is used to create a duplicate message for an existing application then App 2 needs to be changed to consume from a subscription, rather than the original queue.

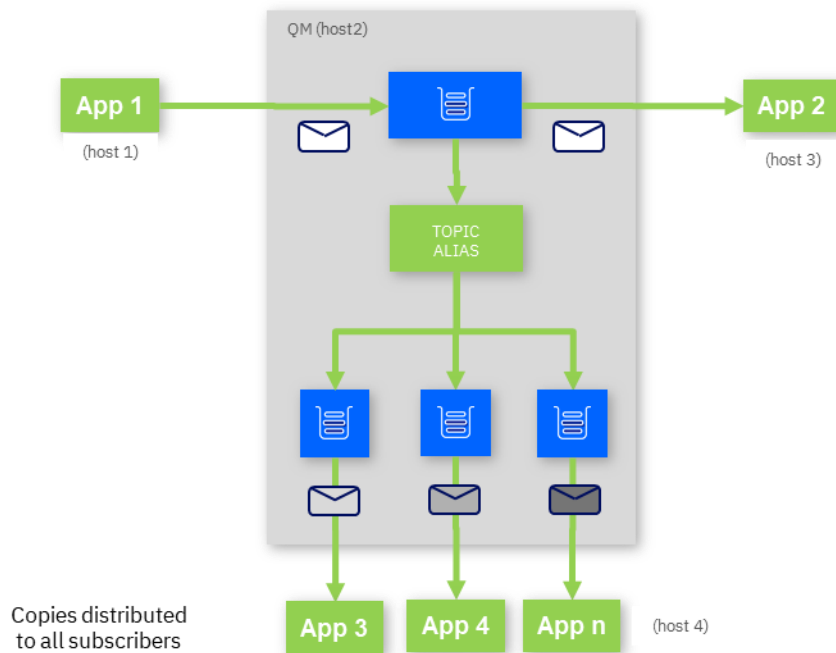


FIGURE 3: 1 TO N DUPLICATE MESSAGES USING STREAMING QUEUES WITH PUB/SUB

Figure 3 shows how multiple duplicates of messages being put onto a queue can be generated by setting STREAMQ to a topic alias and using Pub/Sub to distribute messages to multiple subscribers. The advantages of this solution are that App2 does not need to be switched to a new subscriber queue and the existing applications do not need to be stopped to make the necessary QM changes.

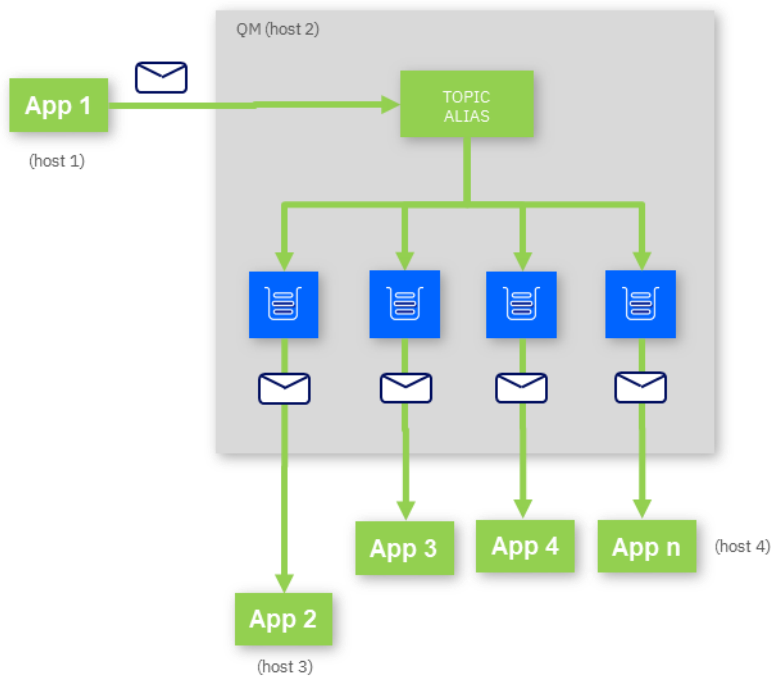


FIGURE 4 : 1 TO N DUPLICATE MESSAGES USING PUB/SUB

Figure 4 shows the topology of using Pub/Sub to generate multiple messages. In the tests the first subscriber (App 2) is on host 3, whilst other subscribers are on host 4. This pattern ensures a common topology where the primary application is on one host and secondary applications, consuming additional (duplicate) messages (for logging or auditing, for example) are on a separate host. If this approach is used to create duplicate messages for an existing application then App 2 needs to be changed to consume from a subscription, rather than the original queue.

5 Scaling the Solutions

In every scenario tested, all getters (including apps draining the stream queues) were started in advance (2 for every queue, ensuring there was always a waiting getter).

Tests start with 1 putter App (App1) and scale up by adding additional groups of putter apps until a limit was approached (CPU or disk bandwidth). Each instance of App1 put to its own queue.

In the simple scenario in Fig 1 for example, the persistent messaging test was scaled up from 1 to 81 putters in groups of 8 putters after the first iteration, as follows:

	Iteration 1	Iteration 2	Iteration 6
# Putters (App1)	1	9	41
# Queues (actively being put to / defined)	1/41	9/41	41/41
# Primary Getters	82 (2 per q)	82 (2 per q)	82 (2 per q)
# Stream Queues (actively being put to / defined)	1/41	9/41	41/41
# Secondary getters (consuming from stream queues)	82 (2 per q)	82 (2 per q)	82 (2 per q)

All Pub/Sub scenarios used unmanaged subscriptions. The subscriber queues each had 2 getters (e.g., two instances each of App3 to App n in Fig 2).

For all tests, getter applications kept up with the putters (App1) so there was no build-up of messages on queues.

The MQI performance test client MQ-CPH was used in all cases (<https://github.com/ibm-messaging/mq-cph>)

6 Results

Results for the tests outlined in section 5 are presented below. For all tests the App1 rate is the measure of how fast a test is running. This is the PUT rate achieved by App1 during the test. The internal message rate will be higher where there are duplicate messages generated. For all tests, the consuming applications kept up with the PUT rates on the original queues and any additional queues (streaming queues or additional unmanaged subscriber queues) so there was no build-up of messages in the queue manager.

Full results for all tests are included in Appendix 2.

6.1 Persistent 2KB Messaging with Single Duplicate Messages.

Figure 5 below shows results for point-to point messaging with and without generating a single duplicate of each message via streaming queues or Pub/Sub. The message rate (total put rate of all App1 instances) is plotted on the y-axis as the number of putters (App1 on the x-axis) is scaled up.

Scenarios tested:

Baseline	No duplicate messages.
1 Duplicate (streaming queues - BESTEF)	Each application queue has STREAMQ set to a local queue for duplicate messages (STREAMQOS = BESTEF).
1 Duplicate (Pub/Sub)	Each application queue is a topic alias with two unmanaged subscriptions.

The total 'internal' messaging rate for the two tests generating duplicates is included below (dashed lines). These will have a total internal message rate twice that of the App1.

Where duplicate messages are produced, these are consumed individually by a 'drainer' application (i.e. each GET from the STREAMQ or additional subscription queue is a distinct transaction, as opposed to batching the GETs which is considered in section 6.2).

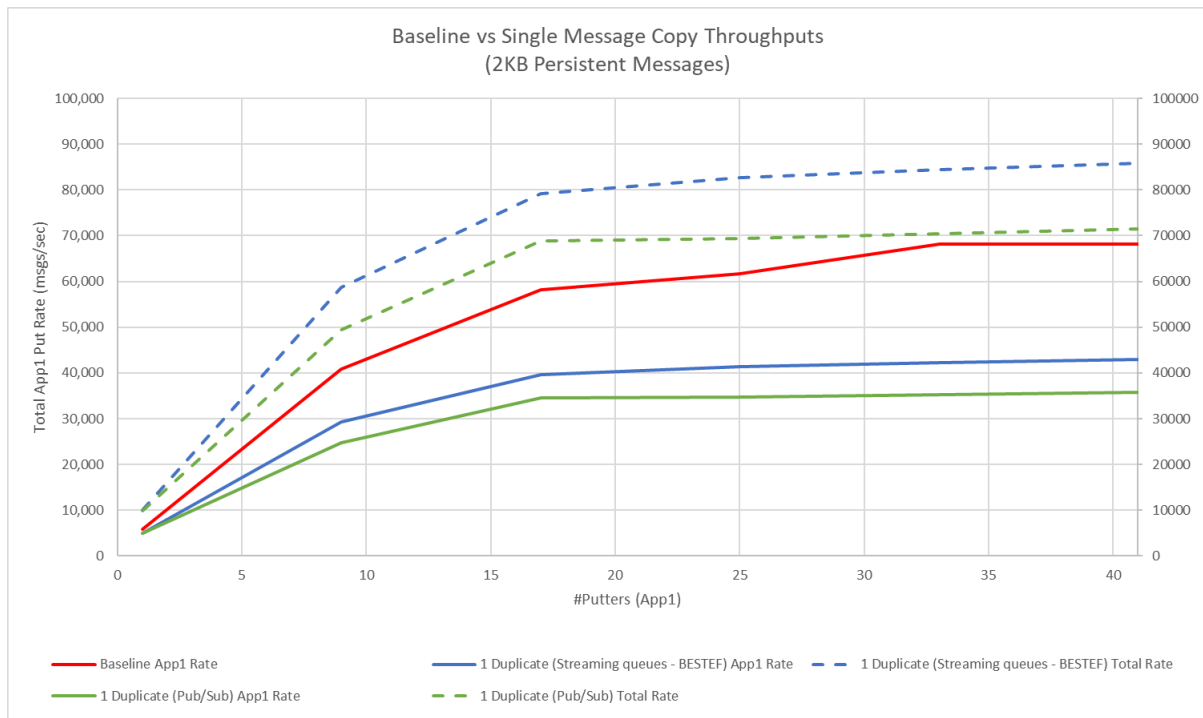


FIGURE 5: BASELINE VS SINGLE COPY THROUGHPUT (2KB PERSISTENT MESSAGING)

Using streaming queues to duplicate each message achieves a slightly higher throughput than using Pub/Sub without the need to modify the original application.

6.2 Batching Duplicate Message Consumption and Overhead of MUSTDUP

Scenarios tested:

1 Duplicate (streaming queues - BESTEF)*

Each application queue has STREAMQ set to a local queue for duplicate messages (STREAMQOS = BESTEF).

1 Duplicate (Streaming queues - drain batch=10) †

Each application queue has STREAMQ set to a local queue for duplicate messages (STREAMQOS = BESTEF). Stream queue drainer apps get in batches of 10.

1 Duplicate (Streaming queues - MUSTDUP)

Each application queue has STREAMQ set to a local queue for duplicate messages (STREAMQOS = MUSTDUP).

* This is the same test as presented in section 6.1 and serves as a baseline here.

† For this test the drainer applications get 10 messages between each commit, rather than a single message per commit as for the rest of the tests.

Messages put by App1 and read by App2 are all done so transactionally, one at a time for the persistent messaging case. This may often be the case for production applications. When reading duplicate messages from stream queues however, it may make sense to batch the reads, which can give a performance gain.

Setting MUSTDUP as the QoS on a stream queue will have a performance impact as there is additional work carried out to ensure the original message is rolled back if the PUT of the duplicate message fails.

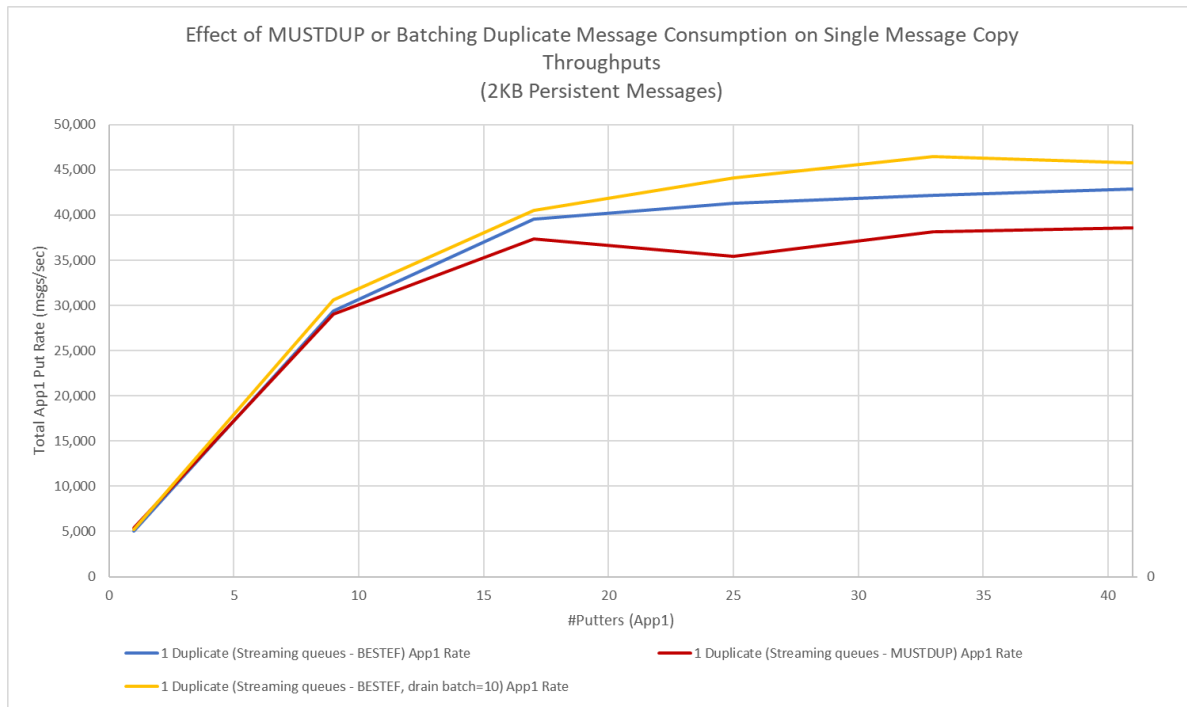


FIGURE 6: EFFECTS OF MUSTDUP OR BATCHING DUPLICATE MESSAGE CONSUMPTION

Results from the tests above are shown in Figure 6.

Batching the duplicate message consumption resulted in a significant increase in overall throughput. This gain will be more or less, depending on other factors, such as message size, performance of the recovery log file system etc. so you should experiment with different values. Bear in mind that large batch sizes may significantly delay the availability to the consuming application and result in more uncommitted data in the recovery log.

Setting MUSTDUP reduced the overall throughput significantly as expected. The overhead of MUSTDUP is significantly more for non-persistent messaging (see section 6.4 below).

6.3 Persistent 2KB Messaging with Multiple Duplicate Messages.

When more than one duplicate of a message is required STREAMQ can be set to point to a topic alias (as in Fig 3 above).

Scenarios tested were:

6 Duplicates (streaming queues – BESTEF, with Pub/Sub)

Each application queue has STREAMQ set to a topic alias with 6 unmanaged subscriptions. (STREAMQOS = BESTEF).

6 Duplicates (Pub/Sub)

Each application queue is a topic alias with 7 unmanaged subscriptions.

Note that there are 7 copies of the message in total for both solutions.

Figure 7 below show the results from this test.

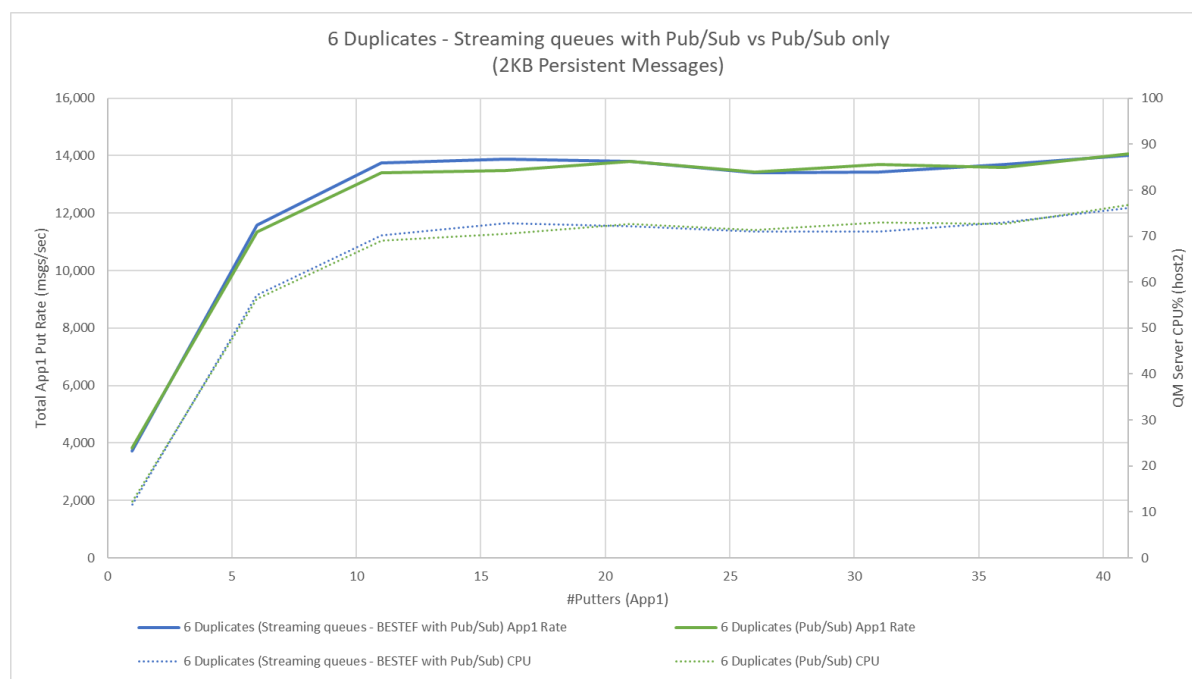


FIGURE 7: 6 DUPLICATES: STREAMING QUEUES WITH PUB/SUB VS PUB/SUB ONLY (2KB PERSISTENT MESSAGING)

Most of the work in both these solutions is being carried out by Pub/Sub in MQ, so unsurprisingly there is little difference between the two approaches in terms of performance, but once again the original applications do not need to be altered to implement the streaming queue approach.

6.4 Peak Rates Achieved for All Scenarios

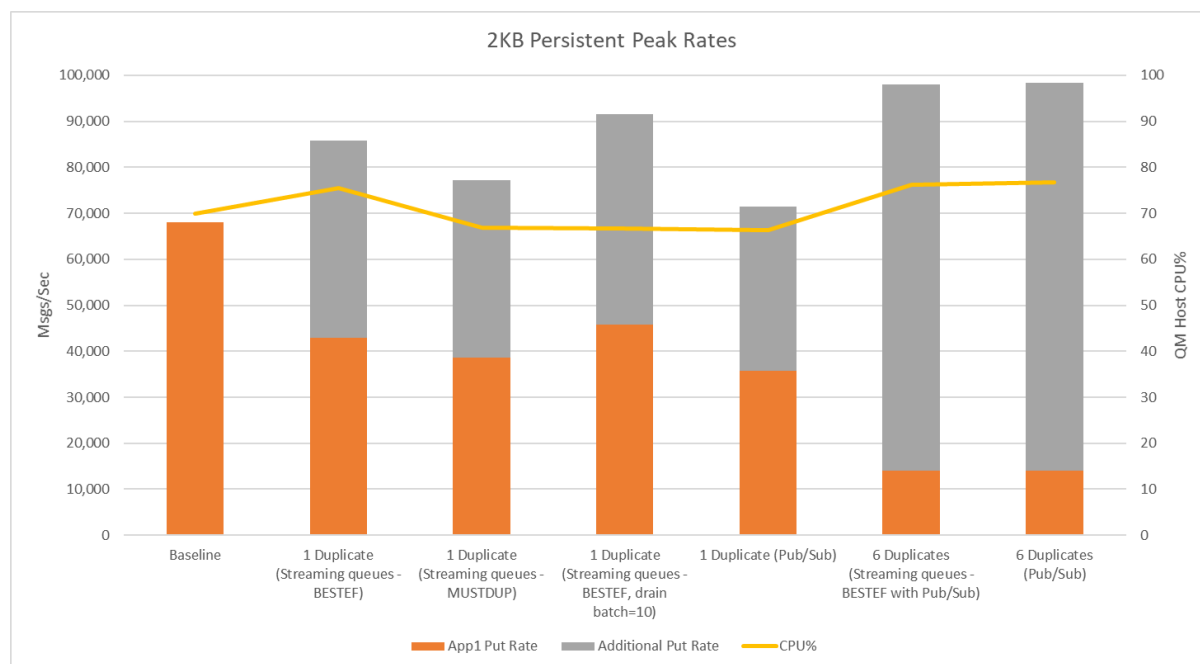


FIGURE 8: PEAK RATES FOR PERSISTENT MESSAGE TESTS

Figure 8 above, shows the peak rates achieved for all the persistent tests presented in the previous sections. The height of each bar represents the peak App1 PUT rate achieved plus the additional internal PUTS caused by message duplication.

Typically, recovery log and locking are the limiting factors for persistent messaging, which is evident here by the CPU consumption not approaching 100% when the peak rate is achieved (compare this to the CPU consumption for non-persistent, below). An existing system already have the CPU capacity to accommodate duplicating messages, but you need to consider the additional data being written to the recovery log for persistent messaging (alongside any additional network bandwidth being consumed by duplicate message consumption).

A tool (MQLDT) is available to assess the recovery log's filesystem performance/capacity (see section 8 below).

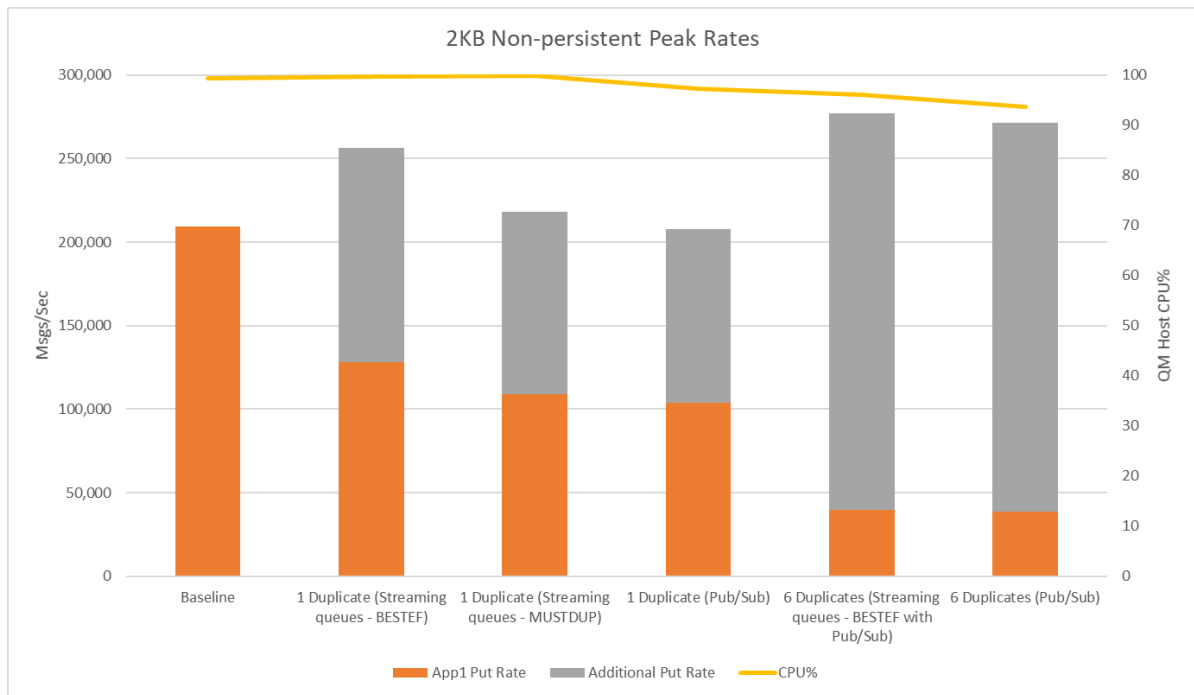


FIGURE 9: PEAK RATES FOR NON-PERSISTENT MESSAGE TESTS

Tests were also run for non-persistent messaging. Figure 9 above shows the peak rates achieved for those tests.

Non-persistent results show a similar pattern to persistent results. Specifying MUSTDUP has a proportionally larger effect on non-persistent messaging (a 15% reduction in throughput compared to BESTEF versus a 10% reduction for persistent messaging in this case).

7 Conclusions

- Streaming queues introduces a new approach to message duplication which does not require changes to existing applications whilst matching or surpassing the performance of previous options (e.g., using Pub/Sub).
- Whilst duplicating each message once will double the internal work rate (assuming the duplicates are also being consumed), the original message rate will not be halved, even when resources on the host are exhausted (see figures 6 & 7).
- Any form of message duplication will involve additional work by the queue manager, so testing and planning for capacity is essential.
 - Additional CPU.
 - Additional data written to the recovery log for persistent messages or if MUSTDUP is specified.
 - Additional network bandwidth utilised/required if remote clients are consuming duplicate messages.
- Specifying MUSTDUP will incur an additional cost, which should be evaluated.
- To minimise additional load on the system and to achieve the best performance, duplicate messages should not be left to build up on the streaming queue. Instead they should be consumed by applications interested in the copy as they arrive.
- It may be possible to optimise the consumption of duplicate messages by batching gets of persistent messages for instance, where this approach was not suitable for the original application.

8 Resources

Streaming queues blog article :	<u>New Streaming Queue feature for MQ 9.2.3</u>
Streaming queues documentation :	<u>https://www.ibm.com/docs/en/ibm-mq/9.2?topic=scenarios-streaming-queues</u>
IBM MQ C Performance Harness (MQ-CPH) :	<u>https://github.com/ibm-messaging/mq-cph</u>
MQ Log Disk Tester (MQLDT) tool:	<u>https://github.com/ibm-messaging/mqldt</u>

Appendix A: Software and Hardware

MQ Server (host 2):

System x3550 M5 -[8869AC1]

CPU 2x14 Cores: Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz

Memory: 128GB

Network: 40Gb via dedicated switch.

O/S : RedHat Enterprise Linux Server V7.9 (Maipo)

MQ : V9.2.3

Primary Client Hosts (host 1 & host3)

System x3550 M5 -[8869AC1]

CPU 2x14 Cores: Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz

Memory: 128GB

Network: 40Gb via dedicated switch.

O/S : RedHat Enterprise Linux Server V7.9 (Maipo)

MQ-CPH Performance Harness with MQ V9.2.3 client libraries.

Secondary Client Host Running Duplicate Message Drainer Apps (host 4)

ThinkSystem SR630 - [7X02CTO1WW]

CPU 2x12: Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz

Memory: 192GB

Network: 40Gb via dedicated switch.

O/S : RedHat Enterprise Linux Server V7.9 (Maipo)

MQ-CPH Performance Harness with MQ V9.2.3 client libraries.

Appendix B: Additional Data

The tables below present the raw data used in the charts throughout this report, including the CPU% of the machine hosting the MQ queue manager. The CPU consumption of client application hosts is not shown as this was negligible, did not present a bottleneck and will not be representative of any specific application in production.

# Clients (App1 Putters)	1	9	17	25	33	41
Baseline App1 Rate	5,872	40,810	58,265	61,644	68,222	68,094
Baseline CPU	4.46	27.74	42.82	55.2	69.33	69.94
1 Duplicate (Streaming queues - BESTEF) App1 Rate	5,019	29,367	39,561	41,332	42,218	42,882
1 Duplicate (Streaming queues - BESTEF) CPU	5.37	45.73	66.92	71.89	72.55	75.43
1 Duplicate (Streaming queues - BESTEF) Total Rate	10,038	58,733	79,121	82,663	84,437	85,765
1 Duplicate (Streaming queues - MUSTDUP) App1 Rate	5,421	29,050	37,371	35,474	38,172	38,585
1 Duplicate (Streaming queues - MUSTDUP) CPU	5.05	44.01	62.28	59.89	64.61	66.92
1 Duplicate (Streaming queues - BESTEF, drain batch=10) App1 Rate	5,205	30,577	40,477	44,106	46,459	45,745
1 Duplicate (Streaming queues - BESTEF, drain batch=10) CPU	4.55	37.24	56.93	62.74	65.92	66.73
1 Duplicate (Pub/Sub) App1 Rate	4,956	24,741	34,462	34,694	35,234	35,755
1 Duplicate (Pub/Sub) CPU	5.21	41.18	62.31	63.64	63.68	66.38
1 Duplicate (Pub/Sub) Total Rate	9,912	49,483	68,925	69,387	70,468	71,510

TABLE 1: SINGLE DUPLICATE RESULTS (SECTIONS 6.1 & 6.2)

# Clients (App1 Putters)	1	6	11	16	21	26	31	36	41
6 Duplicates (Streaming queues - BESTEF with Pub/Sub) App1 Rate	3,712	11,597	13,754	13,891	13,809	13,416	13,423	13,704	14,006
6 Duplicates (Streaming queues - BESTEF with Pub/Sub) CPU	11.6	57.17	70.23	72.8	72.21	70.94	70.96	73	76.18
6 Duplicates (Pub/Sub) App1 Rate	3,817	11,336	13,405	13,494	13,797	13,430	13,704	13,603	14,060
6 Duplicates (Pub/Sub) CPU	12.31	56.27	69.01	70.45	72.68	71.26	73.02	72.63	76.71

TABLE 2: MULTIPLE DUPLICATE RESULTS (SECTION 6.3)

	App1 Put Rate	Additional Put Rate	CPU%
Baseline	68,094	0	69.94
1 Duplicate (Streaming queues - BESTEF)	42,882	42,882	75.43
1 Duplicate (Streaming queues - MUSTDUP)	38,585	38,585	66.92
1 Duplicate (Streaming queues - BESTEF, drain batch=10)	45,745	45,745	66.73
1 Duplicate (Pub/Sub)	35,755	35,755	66.38
6 Duplicates (Streaming queues - BESTEF with Pub/Sub)	14,006	84,033	76.18
6 Duplicates (Pub/Sub)	14,060	84,361	76.71

TABLE 3: PERSISTENT MESSAGING PEAK RATES (SECTION 6.4)

	App1 Put Rate	Additional Put Rate	CPU%
Baseline	209,257	0	99.29
1 Duplicate (Streaming queues - BESTEF)	128,130	128,130	99.7
1 Duplicate (Streaming queues - MUSTDUP)	108,973	108,973	99.85
1 Duplicate (Pub/Sub)	103,840	103,840	97.25
6 Duplicates (Streaming queues - BESTEF with Pub/Sub)	39,568	237,407	96.11
6 Duplicates (Pub/Sub)	38,790	232,740	93.56

TABLE 4: NON-PERSISTENT MESSAGING PEAK RATES (SECTION 6.4)