IBM MQ Performance between z/OS and Linux Using Q Replication processing model

Version 1.0 – February 2018

Tony Sharkey

IBM MQ Performance
IBM UK Laboratories
Hursley Park
Winchester
Hampshire

Notices

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 upon 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 V9.0*. 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 V9.0.

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
- **Intel Corporation:** Intel, Xeon
- **Red Hat:** Red Hat, Red Hat Enterprise Linux

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

In this paper, I will be looking at the performance of MQ in a Q Replication environment between a z/OS queue manager and a Linux system. Much of this content is generally applicable to MQ, but the data presented here is collected on the MQ V9.0.4 CD release on both z/OS and Linux (tested on RedHat EL).

The Q Replication processing model relies on a single task capturing the data from Db2 and putting this data as persistent messages to a "capture" MQ queue. In the traditional model, these messages would then be transported over MQ channels to a remote queue manager and put to the "apply" queue, where they would be browsed by a task which applies the message data to the remote Db2 and subsequently when there are sufficient messages applied, runs a batch process (purge/prune) that destructively gets messages from the apply queue.

For all measurements in this report the "capture" process is run on z/OS with the "apply" and "purge" processes run on Linux.

In our simulation of the Q Replication environment, we use an internal simulation which does not involve Db2 log read/writes and as such is designed to drive to MQ limits.

This paper is split into seven parts:

Part one	- Configuration of z/OS environment and testing model
Part two	- Apply and purge using client directly attached to z/OS queue manager
Part three	- Apply and purge connected via bindings mode to Linux queue manager
Part four	- Latency and the effect of distance on the model
Part five	- Disk performance on distributed machines
Part six	- Tuning the distributed queue manager
Part seven	- Parallel send queues

Part one presents an overview of the different configurations available and measured when using shared queue from z/OS.

Part two presents the results of measurements run when using MQ Clients to perform the apply and purge over server conn channels.

Part three presents the results of measurements run when the apply and purge process running on Linux are connected in bindings mode to a distributed queue manager. These tests use a subset of the z/OS configuration and demonstrate differences in performance between the client mode and the bindings mode measurements.

Part four discusses the effect on distance between the z/OS system and the distributed system for both client and bindings mode configuration. Network latency is injected between the two systems to simulate network traffic flowing over 10KM (cross town) and 50KM (cross city).

Part five discusses the impact of disk I/O performance on the distributed machine as well as offering tools to determining your own disk performance.

Part six offers some tuning guidance for the distributed queue manager to gain best performance in the Q Replication scenario.

Part seven presents the results from using parallel send queues to transport the capture data to the remote queue manager.

Table of Contents

1 Configuration of z/OS environment and testing model

The z/OS system used is configured as described in Appendix A and application programs as described in Appendix B.

For the Q Replication simulation where the apply is performed on a distributed machine, only the capture process is run on z/OS – both the apply and purge are run on distributed.

Unlike previous simulations, this simulation uses shared queue for the capture queue which offers a number of configurations, including the following:

1. Message entirely stored in Coupling Facility (CF).
   Workload is limited to 63KB or less.
2. System is configured with Shared Message Datasets (SMDS).
   SMDS offloads set to default thresholds, i.e.
   - 90% full, all messages larger than ~110 bytes offloaded.
   - 80% full, 4KB and larger messages offloaded.
   - 70% full, all messages of 32KB or larger offloaded.

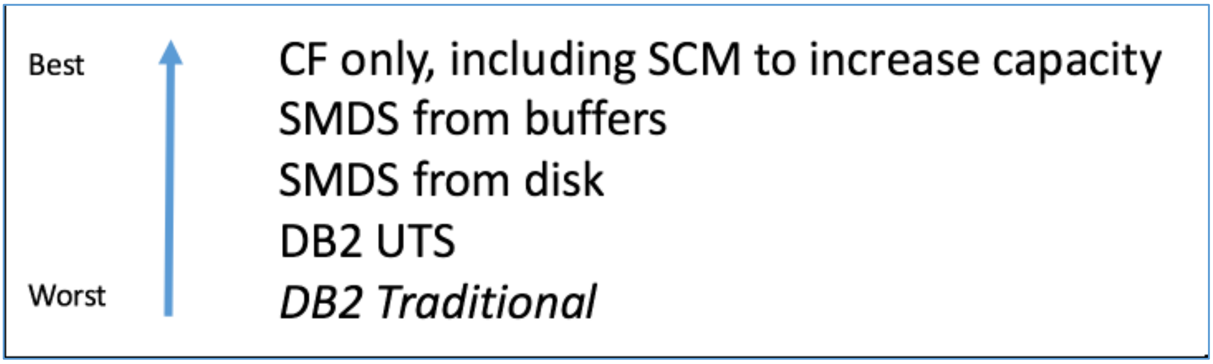
   SMDS configured with 800 buffers of 256KB.
3. System is configured to offload all messages to SMDS.
4. CF has Storage Class Memory (SCM) available to increase capacity.
   Workload limited to 63KB to avoid offload.

   SCM was introduced on zEC12 using Flash Express (SSD on PCIe cards) and was updated on z14 to use Virtual Flash Memory (VFM), which provides better performance by eliminating the I/O adapters – however when the SCM algorithms are keeping pace with the workload, there should be minimal difference in performance.

5. CF has SCM available with SMDS configured at default offload thresholds.
6. System is configured to offload to Db2 Universal Table Spaces (UTS) at default thresholds.
7. System is configured to offload all messages to Db2 UTS.

Each of these configurations offers different benefits, but placing into order of best to worst performing (based on throughput) we see:

Performance based on messaging rate (small messages achieve higher messaging rate[1])



Best

CF only, including SCM to increase capacity

SMDS from buffers

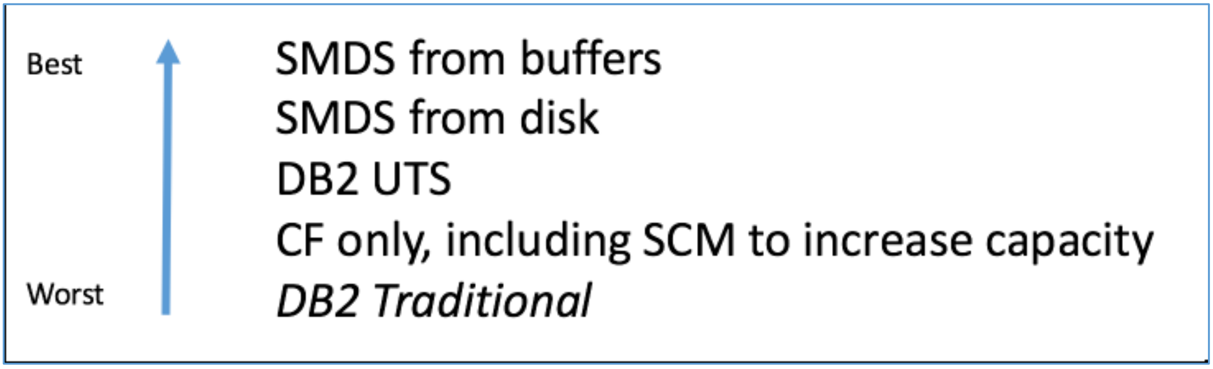
SMDS from disk

DB2 UTS

Worst

DB2 Traditional

Performance based on MB/second throughput (large messages achieve higher volume)



Best

SMDS from buffers

SMDS from disk

DB2 UTS

CF only, including SCM to increase capacity

Worst

DB2 Traditional

The testing uses 2 modes of simulation:

1. **Catch-up**, where the apply queue is allowed to grow in depth until approximately 2GB of data is available to be processed. This model simulates when some delay in the apply system has meant the capture rate exceeds the apply. It also allows us to determine how quickly the apply/purge process can run.
2. **Real-time**, where the apply process runs immediately the first batch of capture messages is available.

The majority of the tests reported use the catch-up model as this focusses on apply/purge performance rather than the rate at which data is processed by z/OS and the network.

The workloads are run with 2KB, 10KB, 32KB, 62KB, 63KB, 100KB and 1MB persistent messages.

Each workload is run using a fixed size payload – the 2KB and 10KB workloads use 5GB of payload and the remaining sizes use approximately 12GB of payload.

[1] Db2 traditional has not been measured in this particular configuration but when testing the Universal Table Space configuration against Db2 traditional, significant performance improvements were observed.

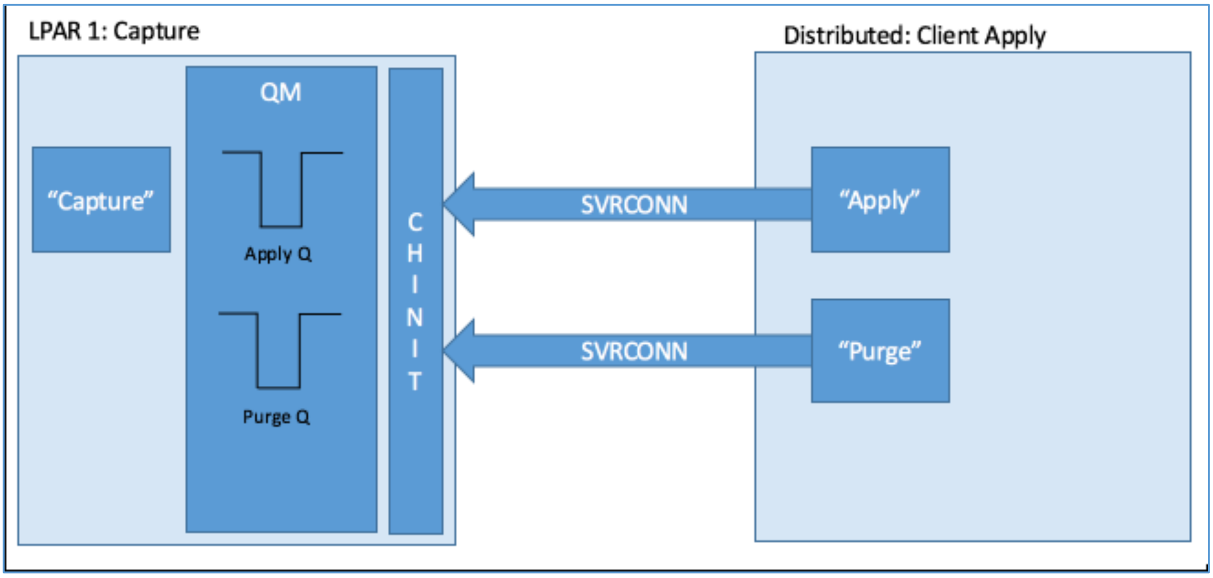
2 Apply and purge using client directly attached to z/OS queue manager

Using a client application to perform the apply and purge results in a model looking as per the following diagram. Note that for the client configuration, the capture and apply queue are the same queue.

For the purposes of these measurements, the SVRCONN channel is configured with SHARECNV(0) as the messages are persistent and cannot benefit from the majority of the function available from channels that support shared conversations.

Performance report MP16 discusses why you may choose shared conversations, which is configured by specifying a non-zero value in the SHARECNV attribute. For persistent workload there are additional flows between client and server due to the use of the asynchronous consume protocol within the channel initiator. This may be a factor to performance if there is significant distance between client and server.

Client-mode apply



The following data is gathered using the "catch-up" processing model.

The capture rate (messages per second) for the different configurations:

Message size	2	10	32	62	63	100	1024
No Offload	6823.9	7003	3339.4	2552.6	N/A	N/A	N/A
SMDS (default)	11910	10963.9	4286.3	3109.3	3114.3	1363	242.1
SMDS (All)	4907.6	4082.3	2700.6	2003.2	3114.3	1363	242.1
SCM	57252.6	30951.8	11239.2	5994.5	N/A	N/A	N/A
SCM (with SMDS)	6910.6	8226.9	3630.2	2689.4	2641.4	1371.5	240.3
Db2 (default)	7008.9	7179	2261.8	2034.3	1994	587.4	127.2
Db2 (all)	829.9	821.6	719	701.5	1994	587.4	127.2

Putting messages such that they are kept in the CF, either by the CF having sufficient capacity or SCM being used for additional capacity ensures the optimum rate of capture.

When the CF is not large enough for the messages and there is no offload facility, the capture process goes into retry processing, which is to wait for 5 seconds and then retry the batch. As is seen from the comparison with the SMDS (default), this can result in a slower capture rate – as demonstrated by the 32KB message capture rate for the "No Offload" configuration, particularly compared to the SCM rate.

It should be noted that in all cases, the rate that messages are captured exceeds the rate at which the client is able to apply/purge those messages, resulting in increasing queue depths until the capture completes its fixed size workload.

The apply process achieved the following rates of messages per second:

Message size	2	10	32	62	63	100	1024
No Offload	3866.39	3630.17	2871.08	2254.89	N/A	N/A	N/A
SMDS (default)	3160.58	3045.09	2058.38	1655.84	1630.82	943.27	175.53
SMDS (All)	2470.92	2181.89	1665.81	1320.05	1630.82	943.27	175.53
SCM	3811.11	3622.19	3012.96	2295.36	N/A	N/A	N/A
SCM (with SMDS)	2747.81	2831.74	1922.27	1553.24	1517.21	945.02	197.66
Db2 (default)	3459.89	3347.57	1947.06	1680.21	1670.99	692.82	145.22
Db2 (all)	1352.55	1360.72	848.7	833.52	1670.99	692.82	145.22

In terms of MB per second, the apply rate table looks as follows:

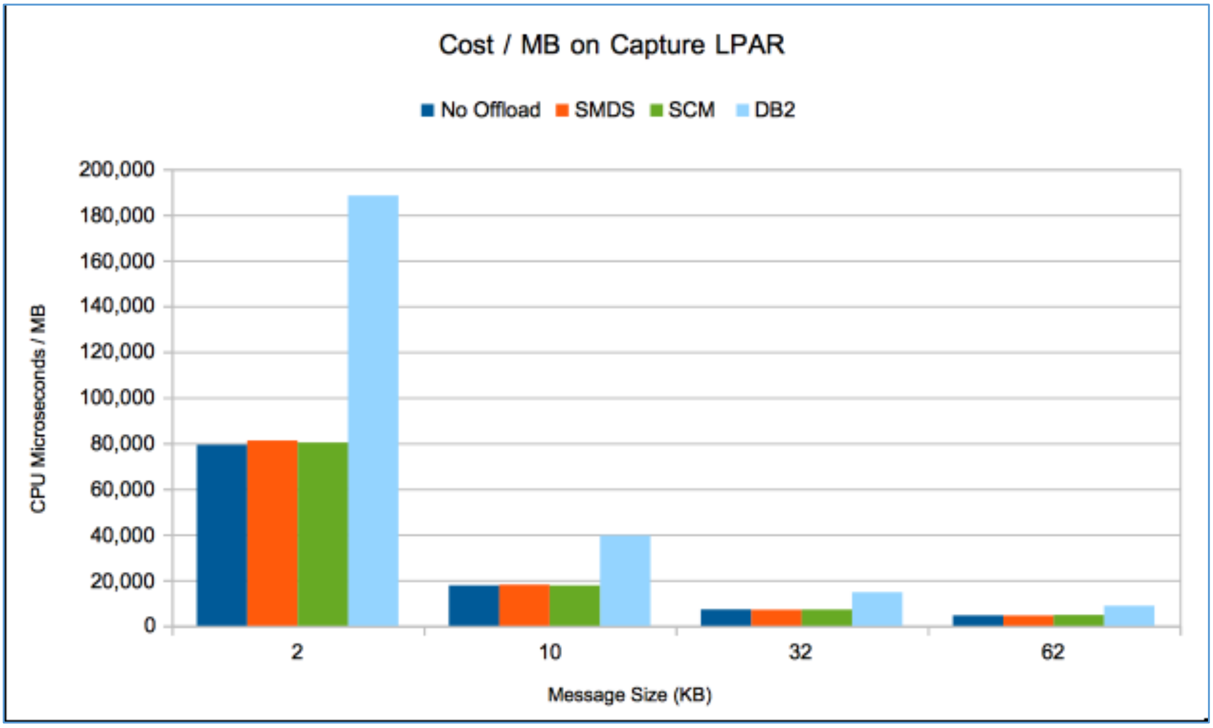
Message size	2	10	32	62	63	100	1024
No Offload	7.55	35.45	89.72	136.53	N/A	N/A	N/A
SMDS (default)	6.17	29.74	64.32	100.26	100.33	92.12	175.53
SMDS (All)	4.83	21.31	52.06	79.92	100.33	92.12	175.53
SCM	7.44	35.37	94.16	138.98	N/A	N/A	N/A
SCM (with SMDS)	5.37	27.65	60.07	94.04	93.34	92.29	197.66
Db2 (default)	6.76	32.69	60.85	101.73	102.81	67.66	145.22
Db2 (all)	2.64	13.29	26.52	50.47	102.81	67.66	145.22

For this model, the purge process is able to keep pace with the apply.

With regards to cost on z/OS, the channel initiator costs are fairly static in terms of transport costs across the configurations.

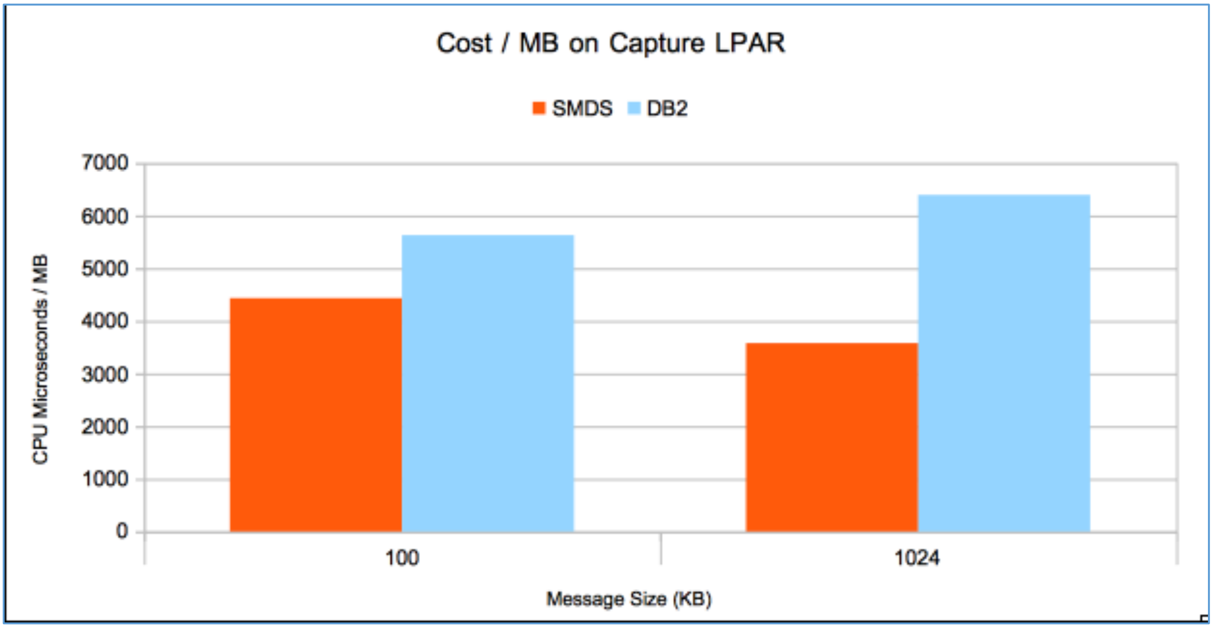
Due to the way costs are charged, the SMDS configuration using larger messages does see an increase in the adaptor costs, whereas the Db2 configuration sees the cost of the Db2 access in the queue manager address space.

Cost per MB for small messages:



Note that SMDS and Db2 data in the small message chart refers to the configuration where all messages are offloaded to either SMDS or Db2.

Cost per MB for large messages:



Note that once the message size exceeds 63KB, there are only 2 configurations measured – SMDS and Db2 UTS. SMDS costs for 100KB messages are equivalent whether the configuration is to offload at default thresholds or to offload everything.

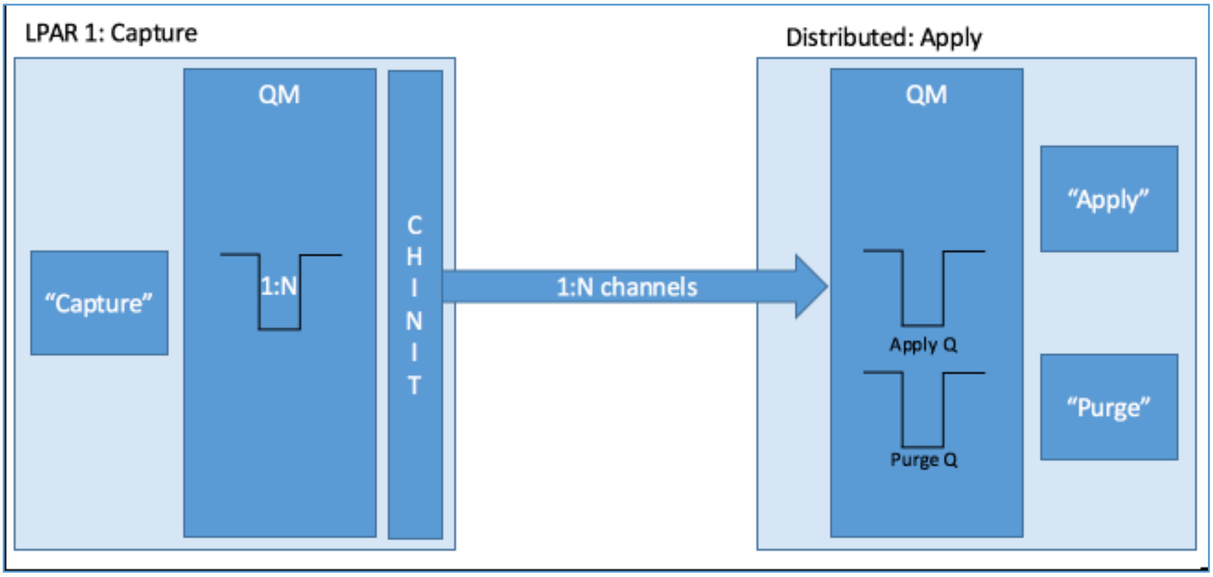
Based on throughput and cost, the best performance on our system is achieved using SMDS for large message offload with sufficient SMDS buffers available to minimize read I/O. In addition SCM storage should be available in the CF to ensure messages smaller than 63KB are available without disk I/O.

3 Apply and purge connected via bindings mode to Linux queue manager

Using a client-side application connecting in bindings mode to a distributed queue manager results in a model looking as per the following diagram.

Using a distributed queue manager for the apply means that each message is logged twice – once on z/OS and once more on distributed, whereas the client apply is only logged once – on z/OS. This means that the ability of the distributed queue manager to log is important and in particular the hardware that the queue manager runs upon should be checked to ensure that it is able to log at the desired rate.

Bindings Apply on Distributed Queue Manager



Note: In the diagram above, the z/OS queue manager is marked with both 1:N queues and 1:N channels – for the purposes of these measurements, N equals 1, i.e. no parallel send queues.

In these measurements, the capture queue manager is configured as CF with SCM available plus SMDS offload, which was the best performing client configuration.

In the catch-up configuration the following information was determined:

Message Size	2	10	32	62	63	100	1024
Capture Rate	11227.50	9006.40	3432.90	2385.70	1896.10	1321.60	257.10
Apply Rate	12871.13	10636.95	2429.49	1340.70	1317.36	803.29	58.72
Apply MB/Sec	25.14	103.88	75.92	81.18	81.05	78.45	58.72

Note that small messages are processed relatively quickly on the apply-side system but larger messages were processed significantly slower. This was due to the load on the I/O subsystem and was determined by running "iostat –ktx". This showed the I/O subsystem was running in excess of 90% busy. The performance report MQIO shows best practise for persistent messaging on Linux queue managers and it may be that on more modern hardware, better apply rates may be observed.

The use of a queue manager to queue manager configuration enables the use of batching of messages over channels. This can significantly reduce the line turn-around time seen when using clients, which have to get each message separately. Historically we have tended to use a BATCHSZ of 200 as this allows several batches of 1MB messages to remain in 31-bit buffer pool, although with the shared queue configuration this size does not have to be a limitation, nor indeed is it a limitation with 64-bit buffer pools.

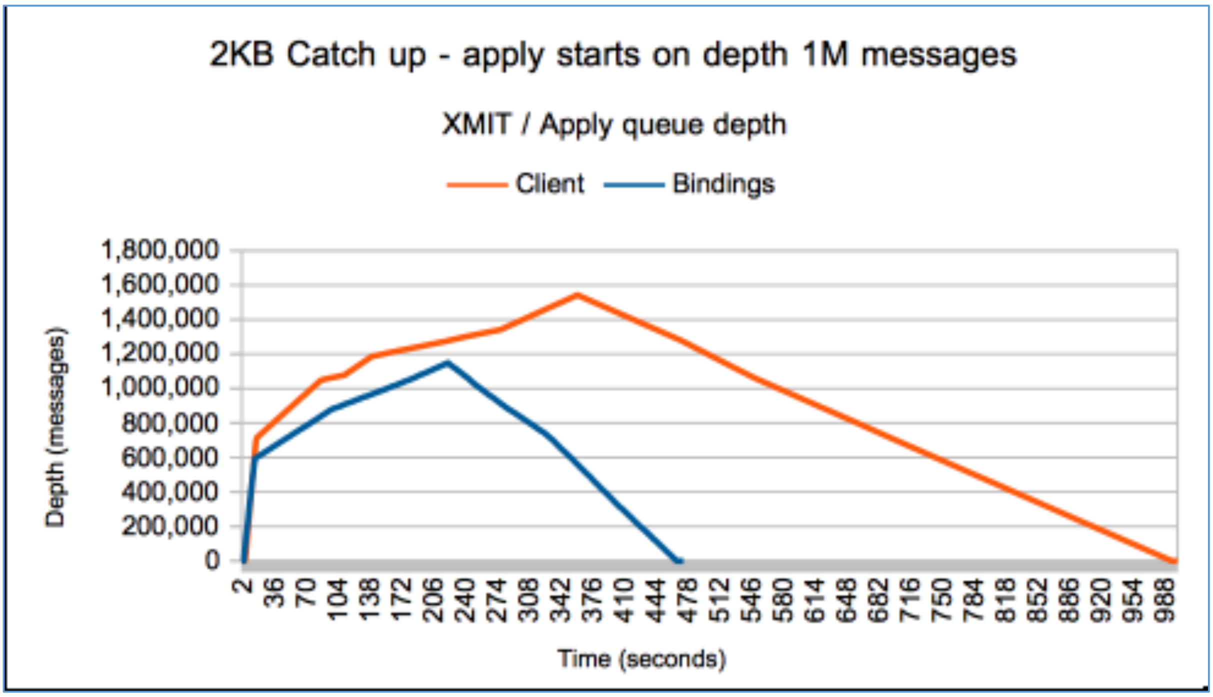
The channel option BATCHLIM represents the limit in kilobytes of the amount of data that can be sent through a channel before taking a sync point. The sync point is taken after the messages that caused the limit to be reached flows across the channel. The default is 5000KB, so if large messages are flowing across the channel, a sync point may be taken after only a small number of message. Therefore, we suggest that BATCHLIM is configured sufficiently large, or disabled, such that BATCHSZ controls the number of messages in a batch.

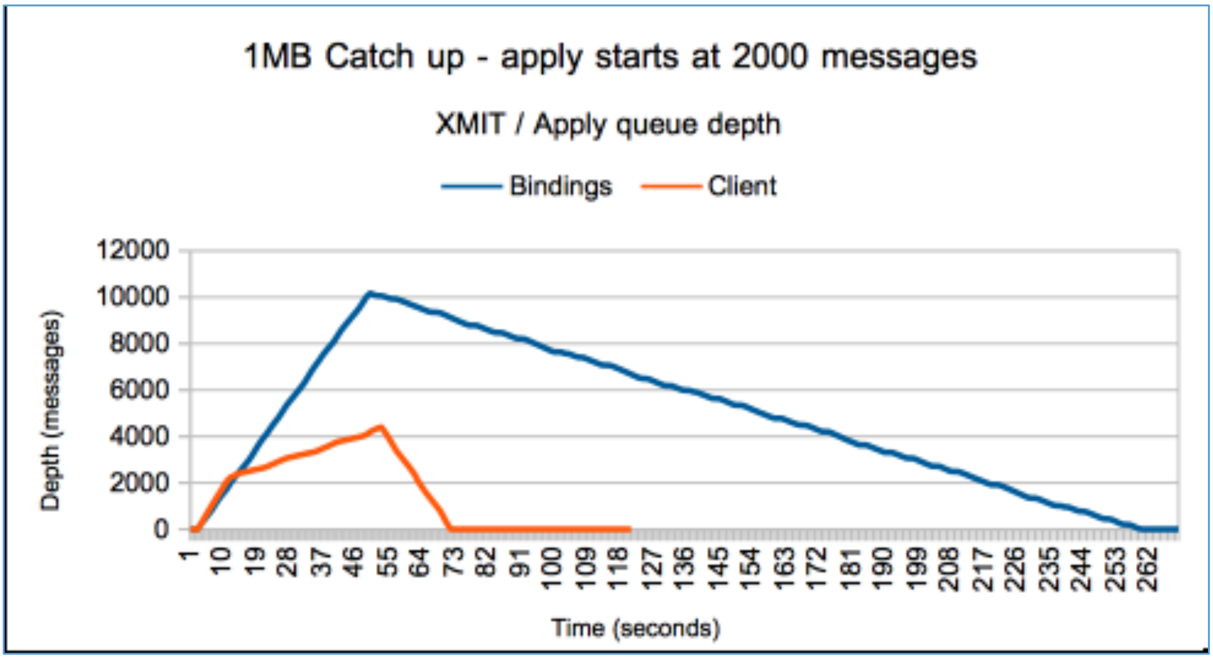
Comparing Client with Bindings configurations

The performance characteristics of the client and bindings models are quite different. For example, the client configuration performs better with large messages – more data per line turnaround, whereas the bindings configuration performs better with smaller messages due to log limits on the distributed machine.

How much difference does the configuration make?

If we compare the XMIT queue depth of the bindings test with the apply queue depth of the client test we can draw some conclusions:





For small messages the bindings model is much more efficient at transporting the message to the target machine, and allows the apply and prune processes to rapidly process the numerous messages. By contrast the client model spends significant time requesting each message for processing.

For large messages, the bindings model is limited by the rate at which the Linux queue managers are able to log the messages arriving, whereas in the client model there is no logging on Linux and each request gives a significant amount of data to process and achieves a much higher rate, significantly reducing the elapsed time for the fixed workload.

Which configuration has the lowest cost on z/OS?

The cost per message on the z/OS systems are shown in the 2 following tables. The queue manager costs are similar for both configurations but the client configuration sees significantly higher cost per message in the channel initiator.

Costs are in CPU microseconds per message.

Client configuration with CF, SCM and SMDS available to z/OS queue manager

Message Size	**2**	**10**	**32**	**62**	**63**	**100**	**1024**
Queue Manager	10.10	16.94	41.23	72.44	72.84	122.30	997.38
Channel Initiator	152.87	157.31	177.76	198.50	200.60	227.53	1237.96
Total	162.97	174.26	218.98	270.94	273.44	349.83	2235.35

Bindings configuration with CF, SCM and SMDS available to z/OS queue manager

Message Size	**2**	**10**	**32**	**62**	**63**	**100**	**1024**
Queue Manager	8.65	17.00	40.93	76.09	81.16	126.05	989.86
Channel Initiator	25.92	36.19	54.03	70.62	71.26	106.60	886.57
Total	34.57	53.19	94.96	146.71	152.42	232.65	1876.42

4 Latency and the effect of distance on the model

In all of these measurements so far, the z/OS LPAR and the distributed machine are located such that there is minimal latency between the two machines, but this is not realistic for a Q Replication environment.

To offer some guidance as to the impact of latency (or distance) on the 2 models, we intercepted the channels (SVRCONN in the case of the client, and Sender-Receiver in the case of the bindings configuration) and added a delay to each flow.

To this end we compare:

- 0 latency
- 10KM latency – cross town – 30 microseconds per flow
- 50KM latency – cross city – 150 microseconds per flow

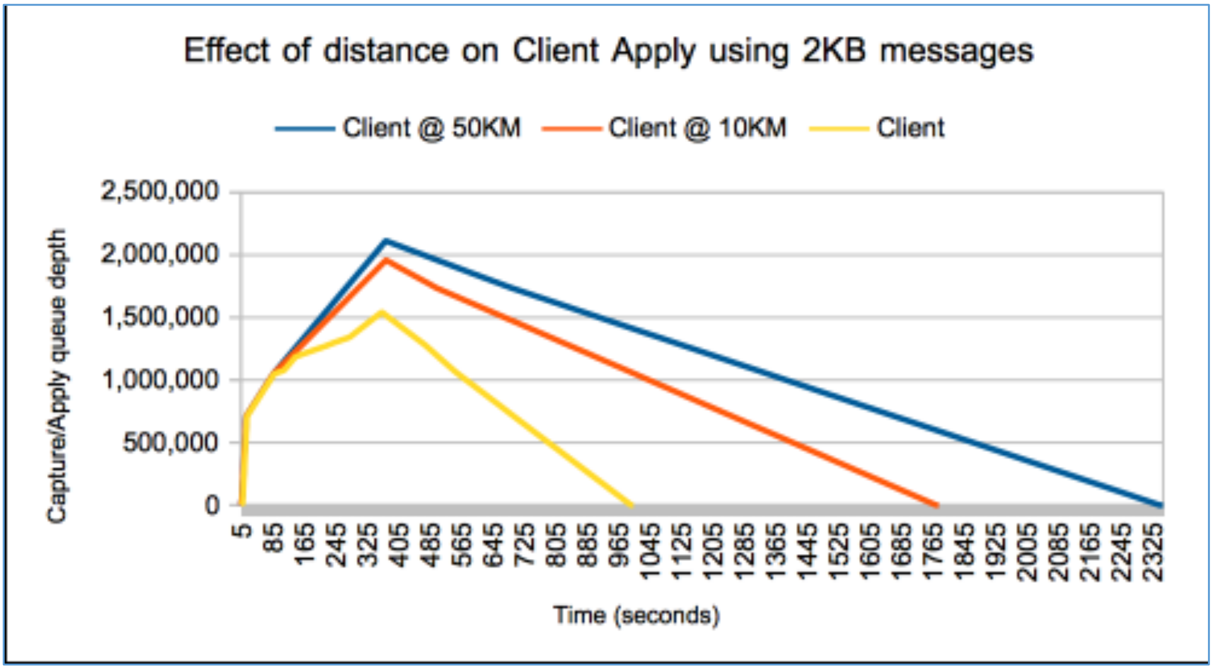
Client configuration

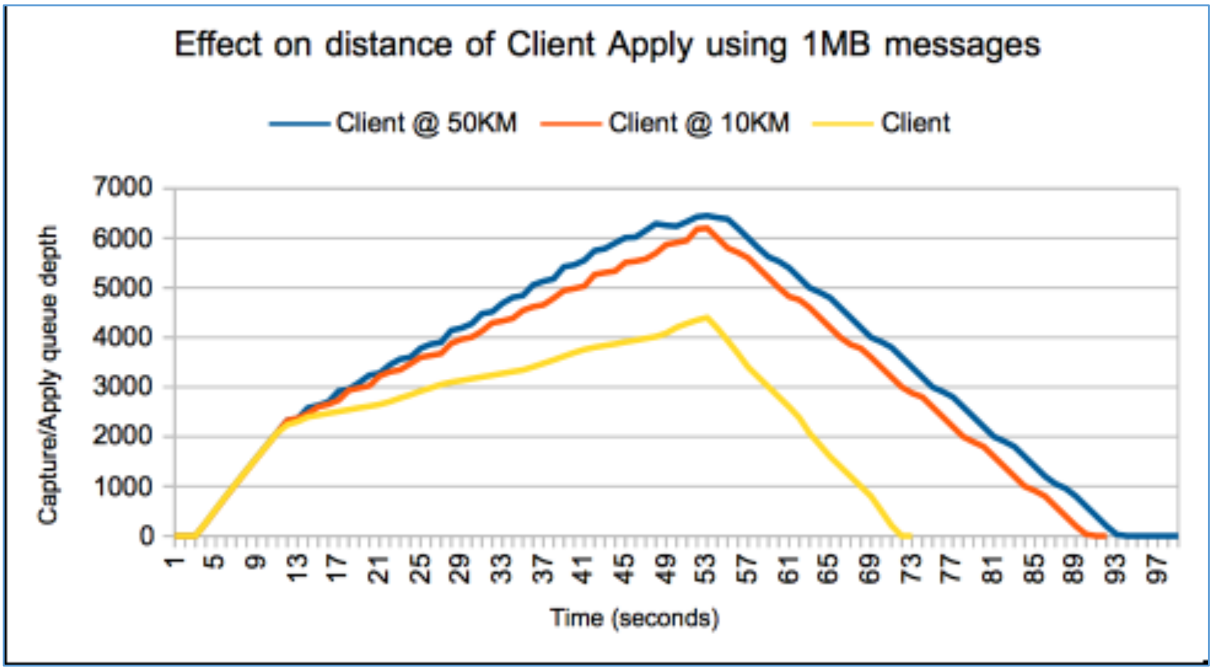
The effects of latency are quite significant for the client configuration. If we consider the rate of apply in MB per second for the range of message sizes:

Message size	**2**	**10**	**32**	**62**	**63**	**100**	**1024**
0 latency	5.31	27.28	59.44	93.23	92.16	91.85	197.24
10KM latency	2.86	14.01	33.88	59.76	60.81	64.24	151.92
50KM latency	2.12	10.21	27.13	45.37	45.85	56.35	146.29

For small messages, adding 50KM of latency reduces the throughput by up to 60%. This impact decreases as the message size grows but even with 1MB messages, the rate drops 25%.

To further illustrate the impact of distance, there are 2 charts showing the capture/apply queue depth:





The wave effect in the 1MB chart shows the impact of the size of the batch. This is more noticeable than in the 2KB messages as there are significantly less messages on the queue.

Bindings configuration

The effects of latency (or distance) on bindings configurations can to a large extent be minimized through the use of BATCHSZ, i.e. sending more data between responses.

The following table shows the rate, in MB per second, at which data is sent from the capture to the apply queue when latency is added.

Message size	2	10	32	62	63	100	1024
0 latency	16.16	60.62	64.73	70.90	69.99	72.46	58.43
10KM latency	16.25	58.01	63.37	67.47	66.26	67.62	52.89
50KM latency	15.58	58.36	64.41	66.43	67.57	68.76	51.02

For this example, the impact is no more than 13% (for large messages) and is much less for smaller messages.

5 Disk performance on distributed machines

Disk performance is of particular interest when using a distributed queue manager in the Q Replication scenario as all messages are persistent.

Disk performance is less of an issue in the Client model, although in the real Q Replication environment there would be logging performed as the Db2 records are inserted.

To measure the utilization of the I/O subsystem on the distributed machine we used the “iostat -ktx” command on 5 second intervals for the period of the measurement. This showed that the volumes used for logging and the queue file were in excess of 90% busy.

Performance paper MQIO offers some guidance on what logging rates may be achieved with MQ on Linux and what actions may be performed to improve the performance.

One of the items identified by the MQIO paper is updating of the QM.ini file to explicitly set “LogBufferPages=4096” to ensure the maximum amount of messages are kept in memory.

Attribute “LogBufferPages” relates to the amount of memory allocated to buffer records for writing, and are specified in units of 4KB pages. The maximum value is 4096 and the use of larger buffers may lead to higher throughput, particularly for larger messages.

The queue manager was configured such that it used circular logs of sufficient size that would not impact the performance of the workload.

What if the workload was non-persistent?

Using non-persistent messages is not the recommended approach for a Q Replication workload but it does offer some indication as to how much impact there is from the logging of persistent messages.

Bindings using persistent messages:

Size (KB)	2	10	32	62	63	100	1024
Apply Rate (MB/Sec)	16.16	60.62	64.73	70.90	69.99	72.46	58.43
Cost / Message	34.57	53.19	94.96	146.71	152.42	232.65	1876.42

Bindings using non-persistent messages:

Size (KB)	2	10	32	62	63	100	1024
Apply Rate (MB/Sec)	33.66	141.84	147.70	197.90	133.43	142.45	214.48
Cost / Message	31.52	41.59	74.85	100.05	71.23	102.32	961.32

By removing the log constraints on the distributed queue manager, the apply rate is improved by 2 to 4 times that of the persistent workload.

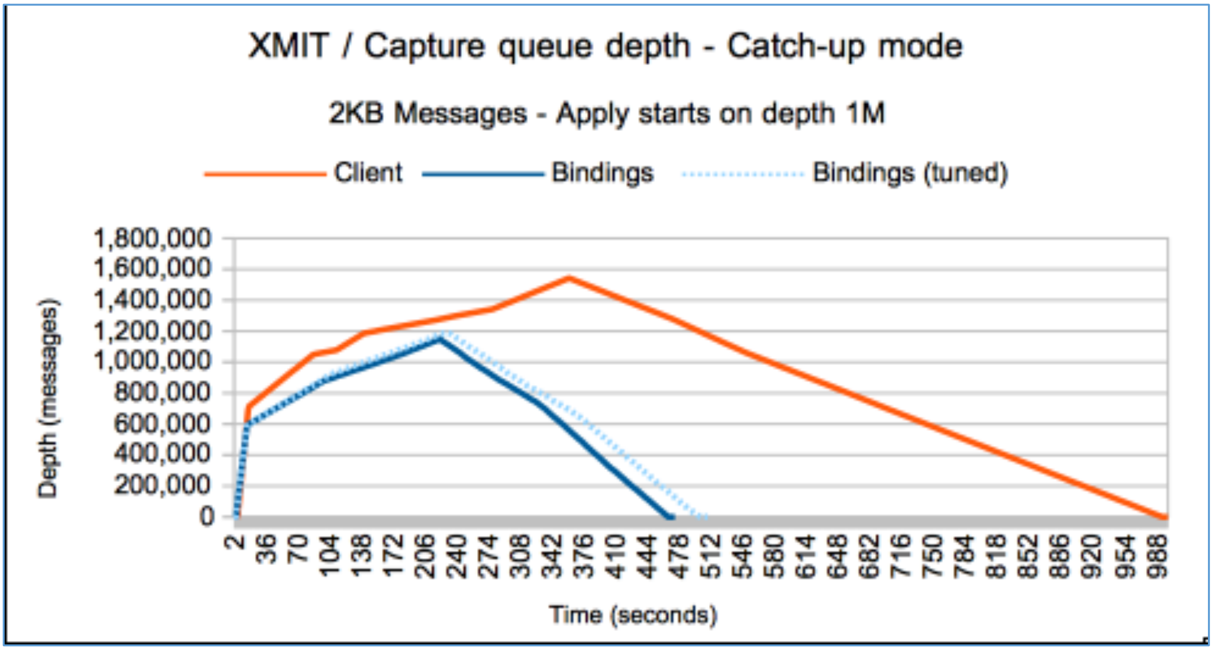
Costs shown are those incurred by the z/OS queue manager and channel initiator and are based on CPU microseconds per message.

6 Tuning the distributed queue manager

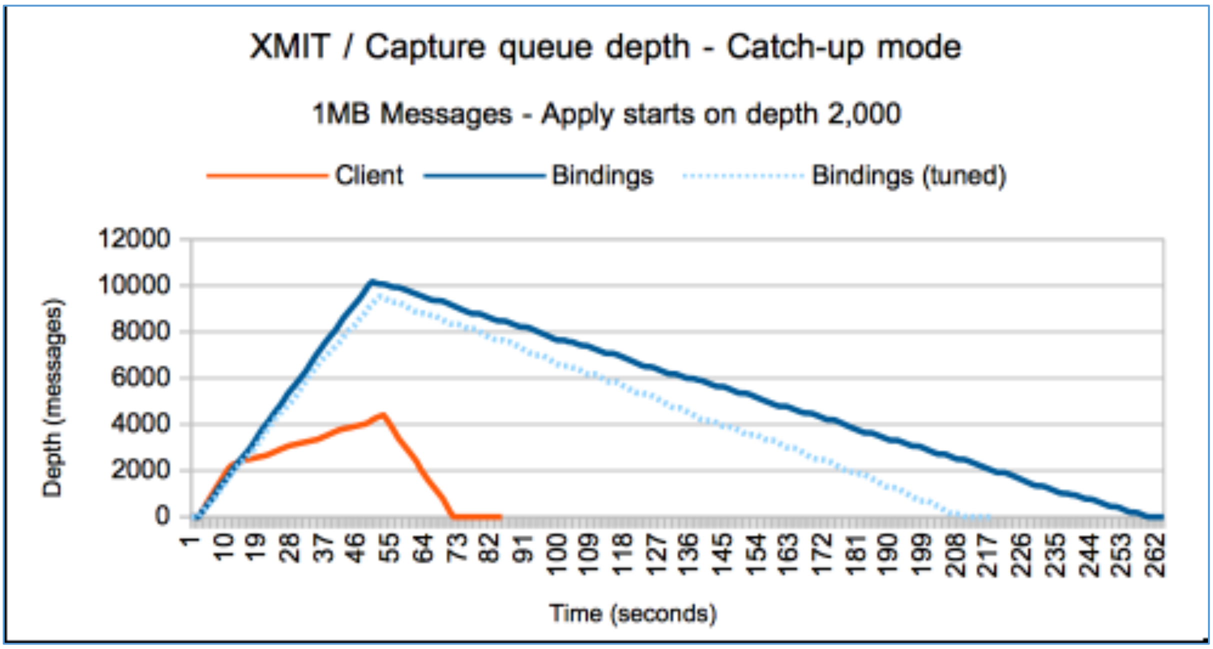
As mentioned in the previous section, it is advised that the QM.ini file is updated to specify “logBufferPages=4096”, and the benefits of this are shown below.

Applying the tuning to the distributed queue manager can make a significant difference, particularly for larger messages as can be seen in the following 2 diagrams. These diagrams show the capture queue depth on z/OS. The rate at which the messages are sent, and therefore put to the distributed queue is indicated by the angle of the line.

Small messages show little benefit – the slight increase in time to process the workload is within typical variation allowance (+/- 6%):



Larger messages can benefit by more than 20% improvement in log rate and a corresponding reduction in run time:



7 Parallel send queues

As mentioned previously, in the client configuration, the capture and apply queue are the same queue, which means that the parallel send queue option is not available.

The use of parallel send queues typically is a mechanism to transport the data more quickly from the capture system to the apply system. If the log rate on the apply system is already the limiting factor, then parallel send queues are unlikely to offer any benefit.

The configuration of the parallel send queues is that there is 1:N capture queues with corresponding channels but only a single target apply queue. This can mean that parallel send queue performance can be impacted by the channels performing at different rates and the messages arriving out of order.

Using a script that issues NETSTAT commands to extract the throughput rate of the sending channels, we can see that parallel send queue configuration improves the total channel throughput rate by 20% for the 2KB workload but offers no benefit to the 1MB workload.

Despite the 2KB workload putting the messages to the apply queue faster when using parallel send queues, the time the apply process took to complete the workload was extended by 30%.

Similarly, the 1MB workload achieved similar channel throughput rates in the 1 or 2 channel measurements but the apply time for the parallel send queue configuration was extended by 24%.

When using parallel send queues, it is possible that apply performance is degraded due to some channels transporting data more quickly and causing data to be stored slightly out of order. This results in deeper apply queues which have to be scanned using CORRELID.

The guidance for performance on distributed MQ is to encourage customers to avoid deep queues due to the extra workload that is placed on the I/O subsystem.

8 Summary

In terms of configuration, both the client and the bindings configuration offer benefits.

Using the bindings configuration there is typically lower cost on z/OS and less impact from network latency. However, there are limitations relating to the rate that the distributed machine can put to a single apply queue and log the persistent messages. There also appears a significant impact from accessing deep queues even when the access uses the optimized path of CORRELID.

The client configuration offers simplicity and achieves good performance with large messages, but does show increased cost on z/OS plus will be impacted more by distance to the queue manager.

In terms of z/OS queue manager configuration, the use of Storage Class Memory is ideal for a Q Replication workload, and offers the ability to keep more data in the Coupling Facility.

The use of Shared Message Data Sets means that there is capacity for up to 16TB of messages to be held in a single structure, allowing for extending outages of the client-side apply/purge. In addition, the MQ “DEFINE CFSTRUCT” command can be configured to specify additional DSBUFS to hold a cached copy of the message. The storage used by DSBUFS is allocated from 64-bit storage and should not impact general MQ performance. The impact of DSBUFS is demonstrated in performance report MP1H.

Alternatively, the Db2 Universal Table Space configuration can be used as this can store up to 128TB of messages but there is a performance trade-off when compared to the SMDS performance.

Appendix A – Test Environment

Measurements were performed using:

IBM MQ Performance Sysplex running on z14 (3906-761) configured thus:

LPAR:

- 16 dedicated general purpose processors with 128 GB of real storage.
- z/OS v2r2.
- Db2 for z/OS version 12 configured for MQ using Universal Table spaces.
- MQ queue managers configured at MQ V9.0.4 and connected to application structures defined with 4GB space.
    - CFLEVEL(5) specified and configured with:
        - No offload of messages greater than 63KB
        - Offload to Storage Class Memory
        - Offload of messages greater than 63KB (or when default thresholds are reached) to Shared Message Data Sets
        - Offload of messages greater than 63KB (or when default thresholds are reached) to Db2 v12 UTS
        - Offload of all messages to SMDS
        - Offload of all messages to Db2 v12 UTS
- MQ queue managers configured with dual logs and dual archives.

Coupling Facility:

- Internal Coupling Facility with 4 dedicated processors
- Coupling Facility running CFCC level 22 service level 00.28
- Dynamic CF dispatching off
- 3 x ICP links between z/OS LPAR and CF.
- Storage Class Memory using Virtual Flash Memory available for some QSG configurations.

DASD:

- FICON Express 16S connected DS8870
- 4 dedicated channel paths
- HYPERPAV enabled
- zHPF disabled for the purposes of the testing

Network:

- 10GbE network configured with minimal hops to distributed machine
- 1GbE network available

Client machine:

Intel Xeon X5660 running at 2.8Ghz with 12 cores and 32GB memory, running Red Hat EL version 7.4.
Machine is connected to 10GbE and 1GbE network.

Appendix B – Test Applications

The Q Replication simulation model used uses 3 application programs, one each for capture, apply and purge. The functions and configurations are described in this section.

Note: MQ on distributed is optimized for get by CORRELID, and as such the measurements have been run with a key of CORRELID.

Capture – program "TESTPUT"

The program TESTPUT is configured to generate messages of a specific size and persistence. The messages are put to one or more (depending on usage of parallel send queues) capture queues with a sequential CORRELATION ID.

The messages are put in batches of configurable size, but historically we have used batches of 200 as this is optimized for private queue using 31-bit buffer pools.

The application can be configured such that when a MQRC 2192 "storage medium full" is returned on the MQPUT, the application goes into retry processing which results in a 5 second wait followed by attempting the MQPUTs again.

When using parallel send queues, the applications puts a batch of messages to queue 1 and commit, then puts a batch of messages to queue 2 and commits, this continues until all of the send queues have 1 batch and then the process is repeated until all required messages are put.

Apply – program "QRAPPLY"

The apply program can be run on z/OS or Linux and is configured to browse by CORRELATION ID to get the messages in the required order.

When using a single capture queue, the messages should always be in the desired order but when using parallel send queues and the messages are transmitted over channels, they can arrive in slightly different order to the sequential ID and therefore a browse-first, browse-next model does not always return the desired message.

The apply program browses a configured number of messages by CORRELATION ID and stores a key to the browsed message. The key may be CORRELID, MSGID or MSGTOKEN. When the desired number of messages has been browsed, the apply program puts a message containing all of these keys to a purge queue for processing by the purge process.

The apply program can be configured to wait until a particular queue depth is reached/exceeded before starting its browse processing. This enables 2 modes of testing:

- Real time
- Catch up – to simulate where there has been some issue on the Apply machine and the queues have increased in depth. The "catch-up" mode tests wait until there is at least 2GB of data on the queue(s).

Purge – program "QRPURGE"

The purge program opens the purge queue and waits for a message. Once a message is successfully gotten, the program processes the table of keys contained in the message data and issues a destructive MQGET with zero length get buffer to delete the messages. When all messages identified in the purge queue message have been processed, the application commits the unit of work.

Appendix C – Reference

MQIO Best practice for persistent messaging on Linux on x86.

MP1H Performance report for WebSphere MQ for z/OS v7.1.0, which contains information on the use and tuning of Shared Message Data Sets.

MP16 MQ for z/OS Capacity and Tuning guide, including information on Coupling Facility performance, Storage Class Memory and SMDS.

Db2 UTS Universal table space support in MQ, samples available from MQ V9.0.4.

CFSizer Sizing the CF structures for MQ.