Exploring the Native HA messaging performance of MQ as part of CP4I deployed on OCP 4.12

Objective

To illustrate the performance capability of MQ when deployed on OpenShift Cloud Platform (OCP) 4.12 as part of Cloud Pak for Integration (CP4I). This report focuses on the performance of Native HA which increases MQ QM availability by replicating log data from the active QM to two replica QM.

Environment

A bare metal setup comprising 3 master nodes, 1 bastion node and 9 worker nodes, although for this test only 3 worker nodes are used (1 for the active QM, 2 for the replica QM). An additional server is used for the client running the containerized test harness.

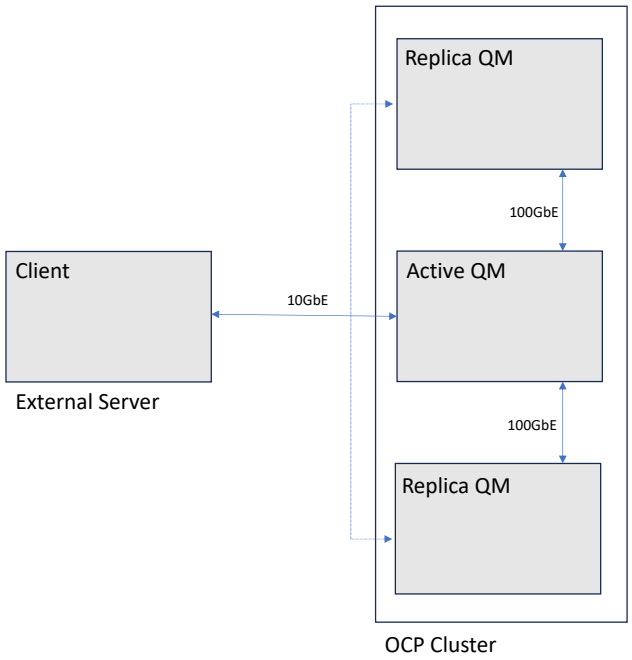


Figure 1 - Test topology

The software versions tested against are included in the following table:

Component	Version
OpenShift Cloud Platform (OCP)	4.12.35
Cloud Pak 4 Integration (CP4I)	2023.2.1
MQ Operator	2.4.5
MQ	9.3.3.0-r2

The persistent volume (PV) for each QM uses a Fibre channel connection to a 50GB LUN on a remote SAN. The access mode required for Native HA storage is RWO (ReadWriteOnce) as each PV is only referenced by a single QM instance. The QM is of type NativeHA.

For more information on the supported configurations, please visit:
https://www.ibm.com/docs/en/ibm-mq/9.3?topic=openshift-release-history-mq-operator

The yaml used to deploy the QM is included in Appendix B. The QM configuration is controlled by mqsc commands in ConfigMaps supplied to the MQ operator during QM creation and can also be found in the same appendix.

The default cluster SDN (Software Defined Network) in this OCP cluster utilizes 1GbE networking. All master and worker nodes are also connected by an additional 10GbE and 100GbE network. The Multus additional network support has been used to allow the client and QM to communicate over the 10GbE network, please see separate guidance (https://github.com/ibm-messaging/mqperf/blob/gh-pages/openshift/configuration.md) on how this was setup. There is a second Multus additional network that utilizes the 100GbE interfaces; this is used for the replication data between the 3 QM instances. To enable the updating of the service endpoints of the 3 QM instances, the multus-service component is required. Please again refer to the above link for further details.

The number of threads supported in a container in this environment has been recently increased (in OCP 4.10) to 4096 threads. This is sufficient for our test scenarios, so no further configuration was required.

The first two sections in this report use a CPU limit of 32 cores for the QM. The last section of the report demonstrates how Native HA performs across QM pods deployed with varying level of CPU resources.

Although in this whitepaper we are comparing Native HA performance with SIQM performance, it should be noted that Native HA removes the need to rely on more resilient replicated storage; adding additional synchronous replication to the data featured here for SIQM would have significantly impacted the throughput presented.

Please see Appendix A for the specification of the hardware used for the tests.

QM Config Changes

Several configuration changes were made via ConfigMaps to improve performance (see Appendix B):

- Enable FASTPATH bindings
- Increase number of MaxChannels/MaxActiveChannels to 5000
- Increased log configuration to 16 primary files (of 16384 4K Pages) although please note that Native HA utilizes replicated logging
- IMGLOGLN set to 25000 which controls the taking of automatic media images
- Increase LogBufferPages to 4096

Scenario

The scenario that will be used in the testing for this whitepaper is the standard requester/responder scenario as featured in our distributed performance reports. For more information on the containerized test harness, please see https://github.com/ibm-messaging/cphtestp/

The MQ client runs in its own container with a fixed number of responders (up to 500) connecting to the QM under test. The test then iterates through an increasing number of client requesters which sends messages across 10 request queues. The responders consume the messages from the request queues and place them on the reply queues where the requester clients obtain their specific reply (via correlation ID) to their original message.

A full round trip as referenced in the results is 2 message puts and 2 message gets; If you are seeking to compare to a base messaging rate of 1 put/1 get, the messaging rates shown should be doubled.

The client runs on hardware external to the OCP cluster and connects to the QM using the address allocated to the QM Pod on the additional 10GbE network and port 1414.

For this investigation, 2KB, 20KB and 200KB Non Persistent and Persistent messages are used. TLS is used for all messaging data flows between the client and QM over the private 10GbE network, and for all MQ replication traffic within the OCP cluster across the additional 100GbE network.

Non Persistent Results

The graph below shows how the MQ QM performs for a 2K message size. Although NP messaging data is not persisted to disk or distributed to the replica Queue Managers; it serves as an illustrative baseline for the messaging performance from remote clients to QM hosted in OCP.

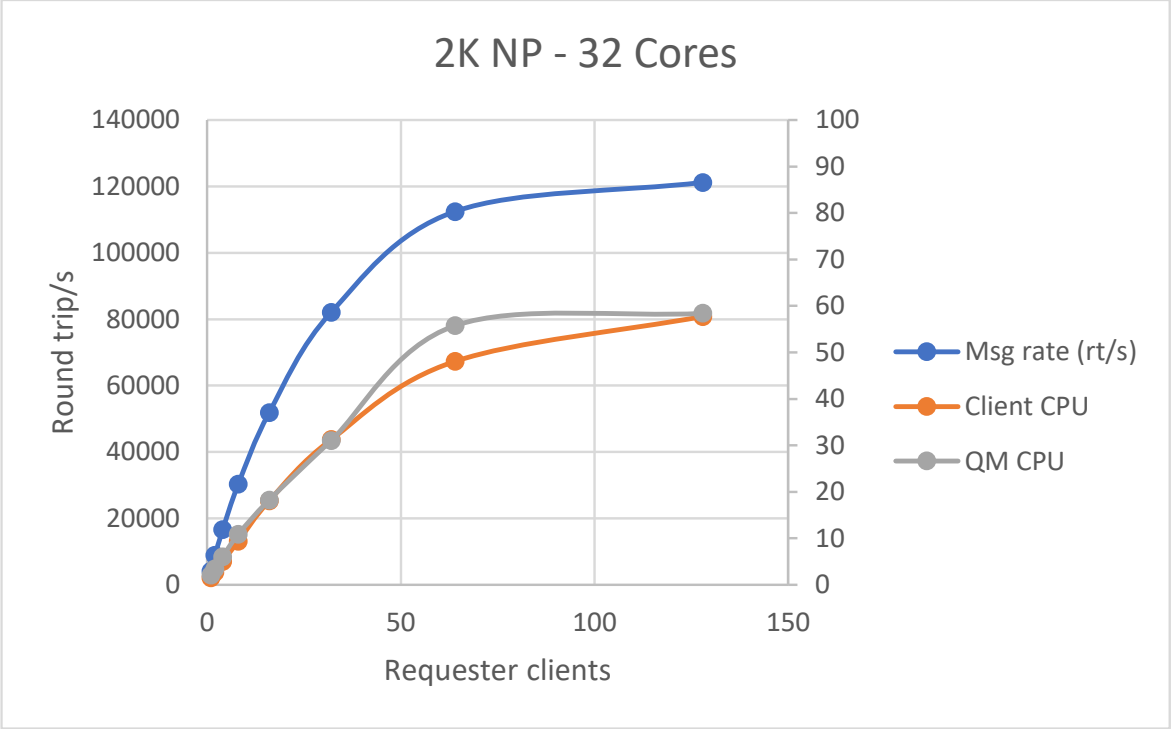


Figure 2 – 2K Non Persistent

Note that the reported CPU is based on the full capacity of the worker node, which in this case is 64 Hyperthreaded cores; a pod restricted to 32 cores would report as having used up to 50% of the available capacity. There are additional pods running on that Node to support the management and configuration of the OCP cluster which is why the maximum reported value is approximately 55%.

The above graph shows that the QM can achieve a peak throughput of over 120,000 round trips/s; and with just 16 requester client threads, the QM can achieve over 50,000 round trips/s.

The graph below shows how the MQ QM performs for a 20K message size.

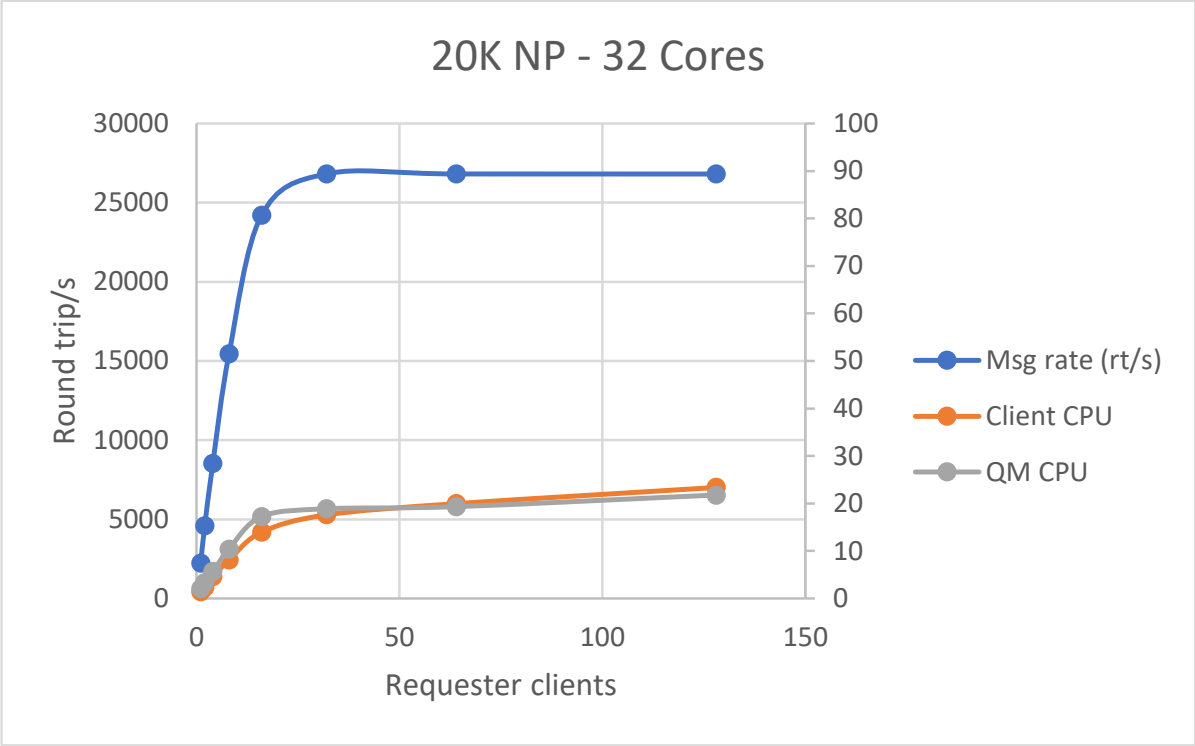


Figure 3 - 20K Non Persistent

The above graph shows that as we increase the message size to 20K, the QM CPU is no longer the limiting factor and we are now limited by our 10GbE workload network at nearly 27,000 round trips/sec, achievable with 32 or more requester clients.

The graph below shows how the MQ QM performs for a 200K message size.

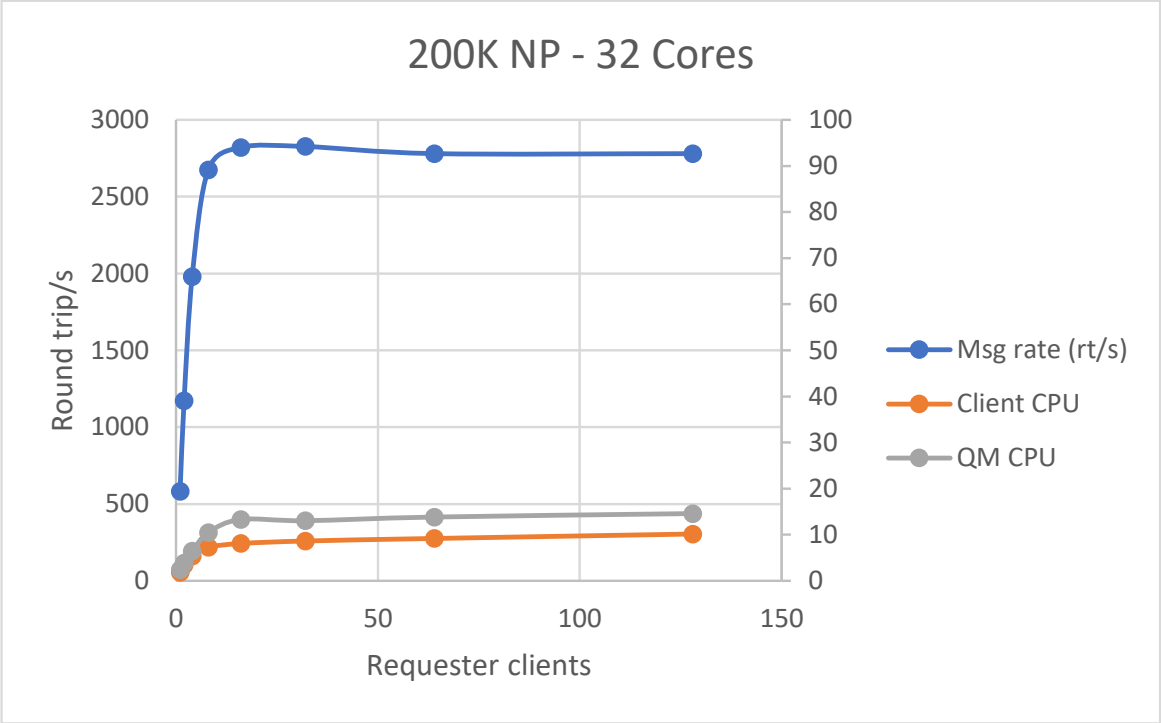


Figure 4 - 200K Non Persistent

The above graph again shows that we are limited by the network when the throughput has reached over 2,800 round trips/s from 16 threads and the QM CPU is ~13% utilised.

Native HA Persistent Results

The graph below shows how the MQ Native HA QM performs for a 2K message size. The results for a Single Instance Queue Manager (SIQM) are also presented.

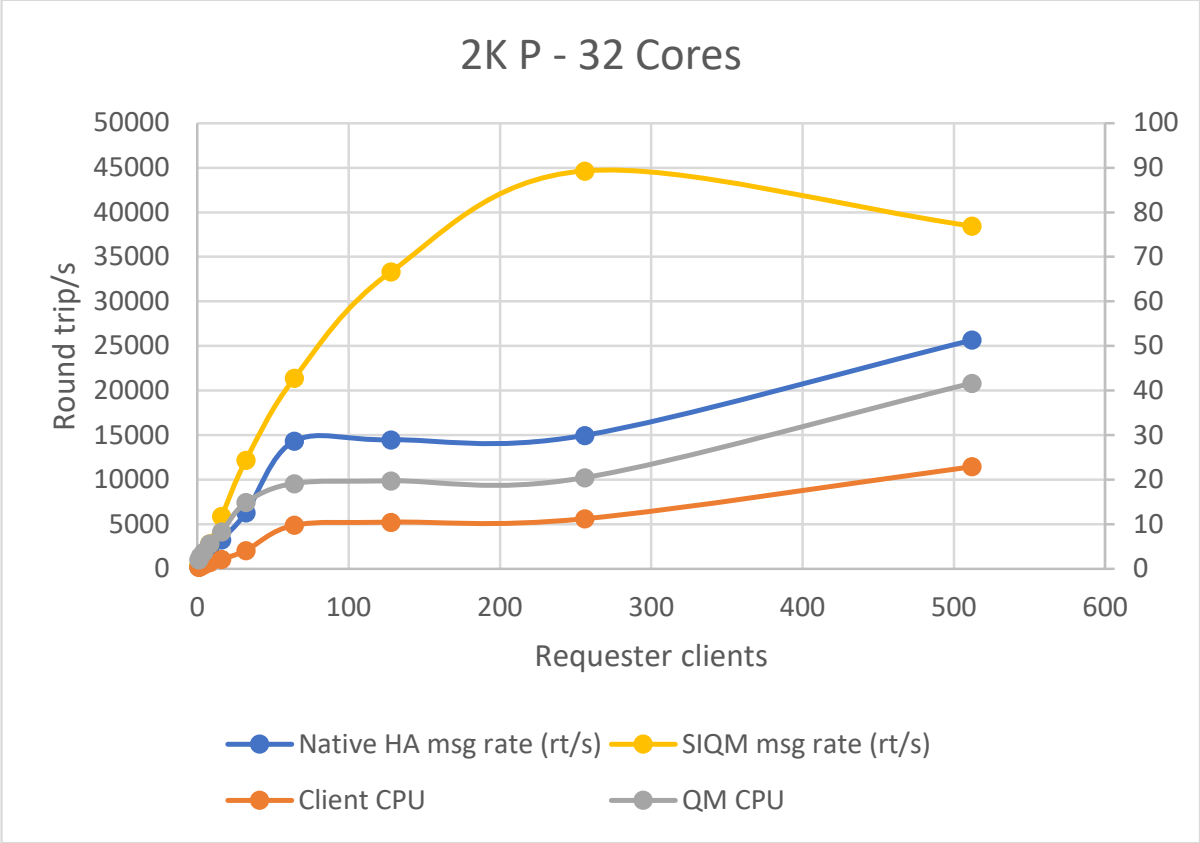


Figure 5 - 2K Persistent

The above graph shows that the Native HA QM can achieve a peak throughput of over 25,000 round trips/s. This is more than 50% of the SIQM performance whilst replicating data across 3 QM.

The CPU of the active QM wasn't saturated in this test and the node utilisation peaks at 42%. The CPU utilisation of the replica QM are much less and both nodes reported less than 10% CPU utilisation. It should be noted that the active and replica QM are deployed with matching CPU and RAM request/limits – as the higher thresholds would be required if the QM were to switchover/failover from the active instance.

The graph below shows how the MQ Native HA QM performs for a 20K message size.

Figure 6 - 20K Persistent

The above graph shows that as we increase the message size to 20K, the Native HA QM is capable of over 5,000 round trips/s. The CPU is not saturated and as the report will show in the following section, this rate is achievable with 16 cores of CPU resources.

The CPU of the active QM peaks at 20%. The CPU utilisation of the replica QM are again much less and both nodes reported less than 5% utilisation.

The graph below shows how the MQ Native HA QM performs for a 200K message size.

Figure 7 - 200K Persistent

The above graph shows that as we increase the message size to 200K, the Native HA QM is capable of over 500 round trips/s. The CPU utilisation is low and as the report will show in the following section, this rate is achievable with 4 cores of CPU resources.

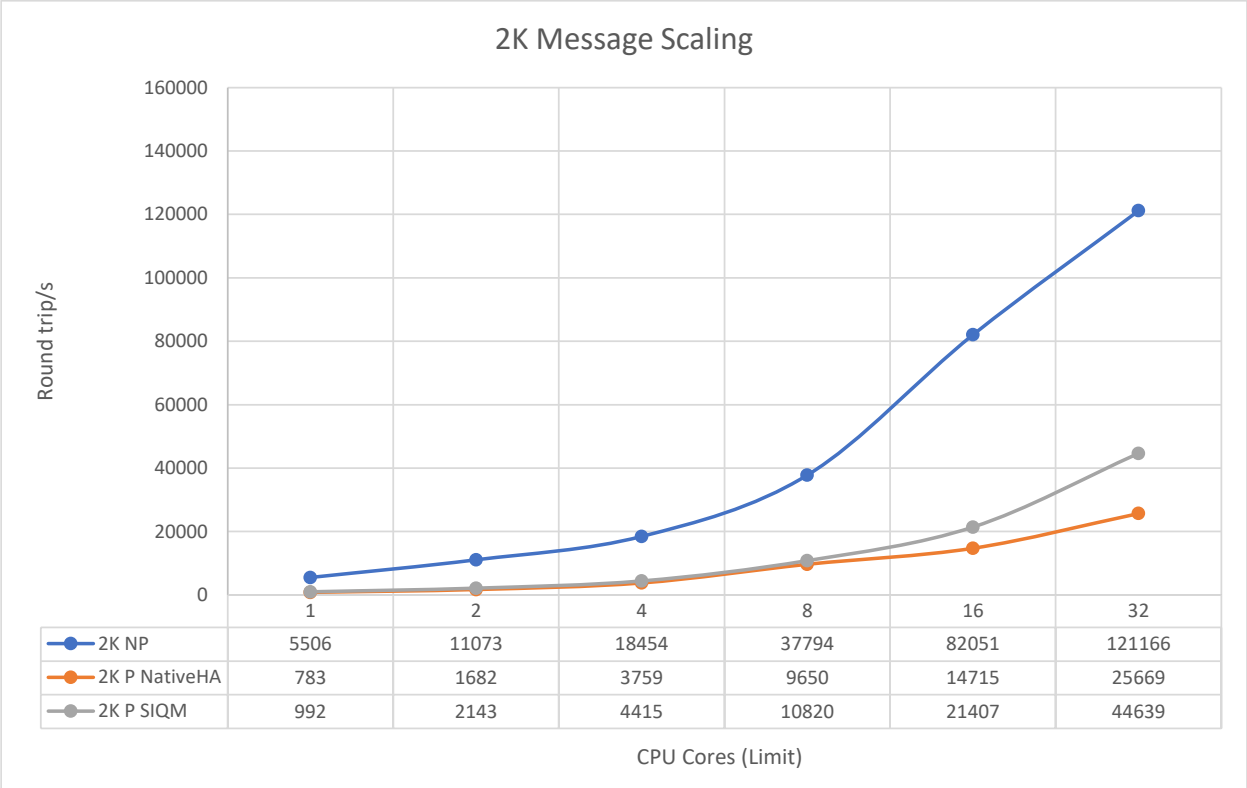
The CPU of the active QM peaks at 8%. The CPU utilisation of the replica QM are again much less and both nodes reported less than 4% utilisation.

Scaling Results

The results presented so far have been with a CPU limit of 32 cores, which is greater than we expect most scenarios to use. 32 cores have been used to show the peak throughput achievable in the specified environment. To illustrate how MQ performs in the OpenShift environment with varying levels of CPU resources, the 2K, 20K and 200K message tests have been run against the MQ QM in multiple CPU configurations and the peak throughput noted.

The active and replica QM are each deployed with the same CPU request/limits; the CPU cores referenced below are allocated to each of the QM.

The graph below shows how the MQ QM scales across varying CPU cores with a 2K message size.



	1	2	4	8	16	32
2K NP	5506	11073	18454	37794	82051	121166
2K P NativeHA	783	1682	3759	9650	14715	25669
2K P SIQM	992	2143	4415	10820	21407	44639

Figure 8 - 2K Scaling

The chart above shows how we can support nearly 800 round trips/s at a single CPU core right up to over 25,000 round trips/s at 32 CPU cores for Native HA Persistent messaging. For messaging against a SIQM the respective values are approximately 1,000 and 44,000 round trips/s. The performance is comparable between Native HA and SIQM up to and including 8 cores.

The graph below shows how the MQ QM scales across varying CPU cores with a 20K message size.

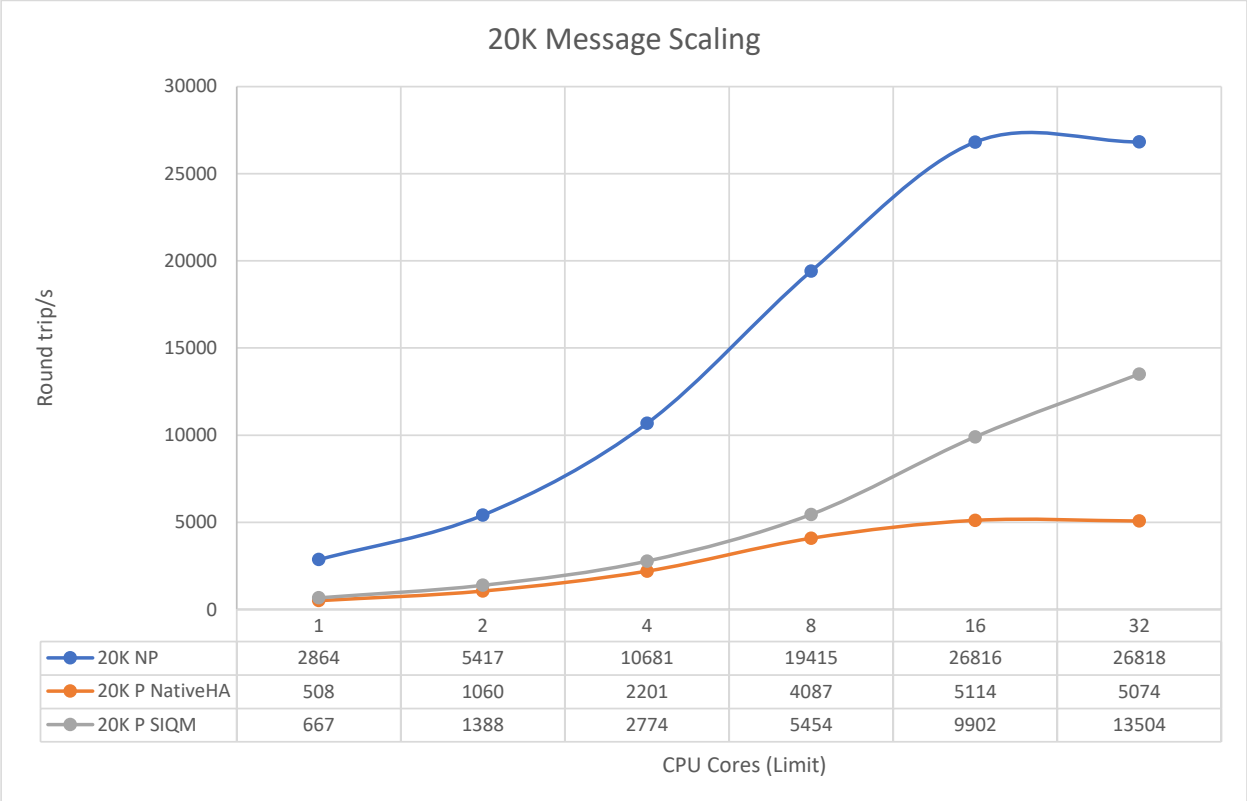


Figure 9 - 20K Scaling

The chart above shows how we can support 500 round trips/s at a single CPU core right up to over 5,000 round trips/s at 16 CPU cores for Native HA Persistent messaging. For messaging against a SIQM the respective values are approximately 650 and 9,900 round trips/s.

The graph below shows how the MQ QM scales across varying CPU cores with a 200K message size.

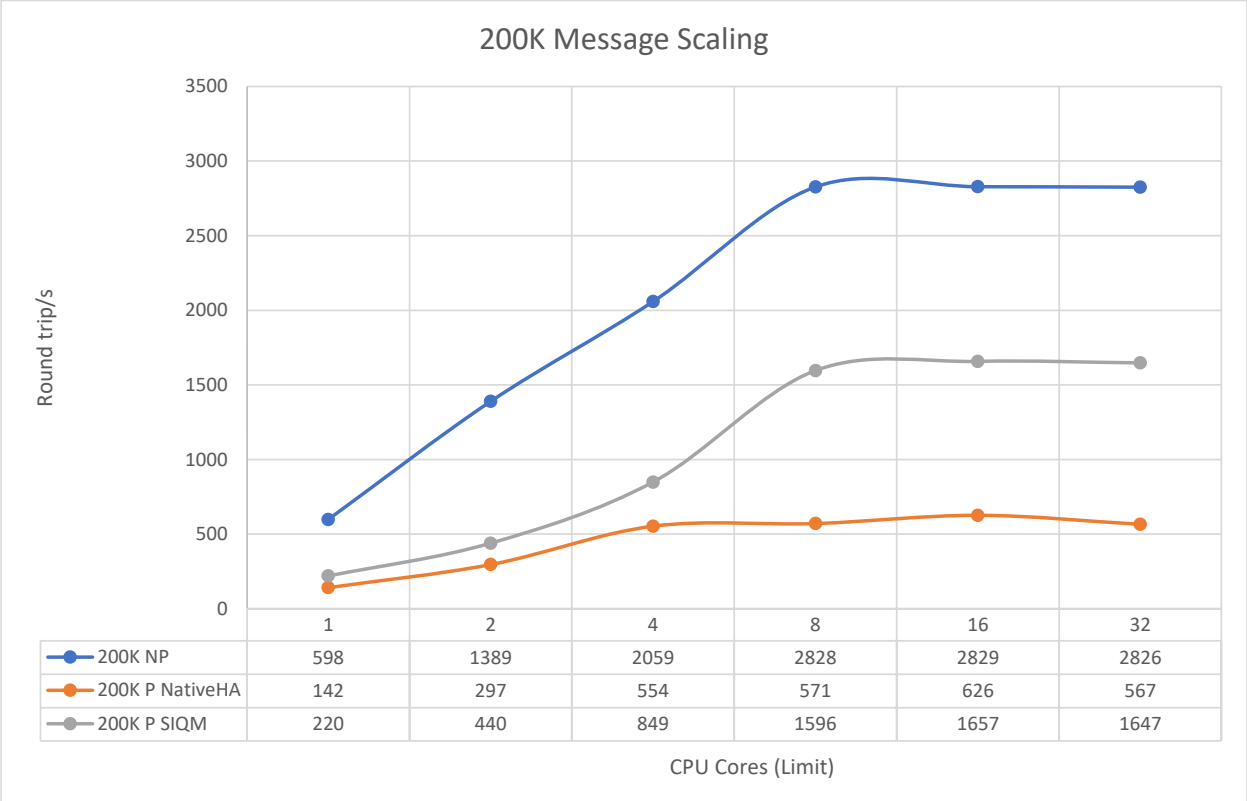


Figure 10 - 200K Scaling

The chart above shows how we can support nearly 150 round trips/s at a single CPU core right up to 550 round trips/s at 4 CPU cores for Native HA Persistent messaging. For messaging against a SIQM the respective values are approximately 220 and 850 round trips/s.

Conclusions

In this whitepaper we have looked at the performance of the Native HA QM in the OpenShift environment when driven by a client outside of the OCP cluster and shown the effect of varying message size, requester clients and CPU cores have on the performance of the QM.

As a comparison point, we have included SIQM data. It should be noted that Native HA has a few advantages over SIQM that are worth highlighting:

- Faster failover in a wide set of failover/switchover scenarios
- Less downtime due to ability to apply rolling updates
- Increased availability across availability zones

Its been shown that Native HA QM can offer both high performance and availability as you deploy QM into the OCP environment and to performance levels comparable with SIQM topologies (particularly at smaller core utilisations).

This data should help you size your solutions to support your intended workload as you deploy your highly available MQ scenarios into the OCP environment.

Appendix A

Hardware specification for Worker Nodes and external client:

System	ThinkSystem SR630
CPU	2x16 Core 2.8Ghz Xeon Gold 6242 Hyperthreaded
RAM	96GB RAM RDIMM TruDDR4 2933MHz
RAID	930-16i 4GB Flash PCI 12Gb RAID Adapter
Disks	800GB SSD (2x400GB) SS530 Performance SAS 12Gbp/s
SAN Connectivity	Dual Port HBA 16Gb
10GbE Network	Dual Port 10GbE Broadcom Network Adapter
100GbE Network	Dual Port 100GbE Mellanox ConnectX-4 Network Adapter

https://lenovopress.com/lp1049-thinksystem-sr630-server-xeon-sp-gen2

Hardware specification for Master, Infrastructure and Bootstrap nodes:

System	ThinkSystem SR530
CPU	1x8 Core 2.1Ghz Xeon Silver 4208 Hyperthreaded
RAM	32GB RAM (2x16GB) RDIMM TruDDR4 2666MHz
RAID	530-8i PCI 12Gb RAID Adapter
Disks	480GB SSD (2x240GB) S4610 Mainstream SATA 6Gbp/s
10GbE Network	Dual Port 10GbE Broadcom Network Adapter

https://lenovopress.com/lp1045-thinksystem-sr530-server-xeon-sp-gen2

Appendix B

QM Yaml:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: perf0
  annotations:
    k8s.v1.cni.cncf.io/networks: default/tengig,default/hundredgig
    k8s.v1.cni.cncf.io/service-network: default/hundredgig
  labels:
    service.kubernetes.io/service-proxy-name: multus-proxy
spec:
  version: 9.3.3.0-r2
  license:
    accept: true
    license: L-YBXJ-ADJNSM
    use: Production
  pki:
    keys:
    - name: default
      secret:
        secretName: mqcert
        items:
          - tls.key
          - tls.crt
    trust:
    - name: app
      secret:
        secretName: mqcert
        items:
          - app.crt
  web:
    enabled: false
  queueManager:
    name: "PERF0"
    availability:
      type: NativeHA
      tls:
        secretName: mqcert-internal
        cipherSpec: ANY_TLS12
    storage:
      defaultClass: san
      queueManager:
    mqsc:
    - configMap:
        name: perf-mqsc-ini
        items:
        - disable-tls.mqsc
        - nha.mqsc
        - queues.mqsc
    ini:
    - configMap:
        name: perf-mqsc-ini
        items:
        - fastpath.ini
    recoveryLogs:
      logFilePages: 16384
    resources:
      limits:
        cpu: '32'
        memory: 16Gi
```

```
      requests:
        cpu: '32'
        memory: 16Gi
  securityContext:
    initVolumeAsRoot: true
  template:
    pod:
      containers:
       - name: qmgr
         env:
         - name: MQSNOAUT
           value: "yes"
```

MQSC ConfigMap yaml:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: perf-mqsc-ini
data:
  disable.mqsc: |
    alter qmgr chlauth(disabled)
    alter qmgr maxmsgl(104857600)
    alter channel(SYSTEM.DEF.SVRCONN) chltype(SVRCONN) sharecnv(1) mcauser('mqm')
maxmsgl(104857600) SSLCAUTH(OPTIONAL) SSLCIPH('')
    alter qlocal(SYSTEM.DEFAULT.LOCAL.QUEUE) maxmsgl(104857600)
    alter qmodel(SYSTEM.DEFAULT.MODEL.QUEUE) maxmsgl(104857600)
    alter qmodel(system.jms.tempq.model) maxmsgl(104857600)
    alter qlocal(system.dead.letter.queue) maxmsgl(104857600)
    alter authinfo(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) authtype(IDPWOS) chckclnt(OPTIONAL)
    refresh security(*) type(CONNAUTH)
  disable-tls.mqsc: |
    alter qmgr chlauth(disabled)
    alter qmgr maxmsgl(104857600)
    alter channel(SYSTEM.DEF.SVRCONN) chltype(SVRCONN) sharecnv(1) mcauser('mqm')
maxmsgl(104857600) SSLCAUTH(REQUIRED) SSLCIPH('ANY_TLS12_OR_HIGHER')
    alter qlocal(SYSTEM.DEFAULT.LOCAL.QUEUE) maxmsgl(104857600)
    alter qmodel(SYSTEM.DEFAULT.MODEL.QUEUE) maxmsgl(104857600)
    alter qmodel(system.jms.tempq.model) maxmsgl(104857600)
    alter qlocal(system.dead.letter.queue) maxmsgl(104857600)
    alter authinfo(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) authtype(IDPWOS) chckclnt(OPTIONAL)
    refresh security(*) type(CONNAUTH)
  nha.mqsc: |
    alter qmgr IMGLOGLN(25000)
  queues.mqsc: |
    define qlocal(queue) maxdepth(5000) replace
    define qlocal(request) maxdepth(5000) replace
    define qlocal(reply) maxdepth(5000) replace
    define qlocal(request1) maxdepth(5000) replace
    define qlocal(request2) maxdepth(5000) replace
    define qlocal(request3) maxdepth(5000) replace
    define qlocal(request4) maxdepth(5000) replace
    define qlocal(request5) maxdepth(5000) replace
    define qlocal(request6) maxdepth(5000) replace
    define qlocal(request7) maxdepth(5000) replace
    define qlocal(request8) maxdepth(5000) replace
    define qlocal(request9) maxdepth(5000) replace
    define qlocal(request10) maxdepth(5000) replace
    define qlocal(request11) maxdepth(5000) replace
    define qlocal(request12) maxdepth(5000) replace
    define qlocal(request13) maxdepth(5000) replace
    define qlocal(request14) maxdepth(5000) replace
```

```
  define qlocal(request15) maxdepth(5000) replace
  define qlocal(request16) maxdepth(5000) replace
  define qlocal(request17) maxdepth(5000) replace
  define qlocal(request18) maxdepth(5000) replace
  define qlocal(request19) maxdepth(5000) replace
  define qlocal(request20) maxdepth(5000) replace
  define qlocal(reply1) maxdepth(5000) replace
  define qlocal(reply2) maxdepth(5000) replace
  define qlocal(reply3) maxdepth(5000) replace
  define qlocal(reply4) maxdepth(5000) replace
  define qlocal(reply5) maxdepth(5000) replace
  define qlocal(reply6) maxdepth(5000) replace
  define qlocal(reply7) maxdepth(5000) replace
  define qlocal(reply8) maxdepth(5000) replace
  define qlocal(reply9) maxdepth(5000) replace
  define qlocal(reply10) maxdepth(5000) replace
  define qlocal(reply11) maxdepth(5000) replace
  define qlocal(reply12) maxdepth(5000) replace
  define qlocal(reply13) maxdepth(5000) replace
  define qlocal(reply14) maxdepth(5000) replace
  define qlocal(reply15) maxdepth(5000) replace
  define qlocal(reply16) maxdepth(5000) replace
  define qlocal(reply17) maxdepth(5000) replace
  define qlocal(reply18) maxdepth(5000) replace
  define qlocal(reply19) maxdepth(5000) replace
  define qlocal(reply20) maxdepth(5000) replace
fastpath.ini: |
  Channels:
  MQIBindType=FASTPATH
  MaxActiveChannels=5000
  MaxChannels=5000
  Log:
  LogPrimaryFiles=16
  LogSecondaryFiles=2
  LogBufferPages=4096
  TuningParameters:
  DefaultPQBufferSize=10485760
  DefaultQBufferSize=10485760
```