Exploring the internal messaging performance of MQ as part of CP4I deployed on OCP 4.10

Objective

To illustrate the performance capability of MQ when deployed on OpenShift Cloud Platform (OCP) 4.10 as part of Cloud Pak for Integration (CP4I).

Environment

A bare metal setup comprising 3 master nodes, 1 bastion node and 9 worker nodes, although for this test only 2 worker nodes are used (1 for the Queue Manager (QM) and 1 for the MQ Client).

OpenShift Cloud Platform Version: 4.10.26

CP4I Version: 2022.2.1

MQ Operator: 2.06

MQ Version: 9.3.0.0-r3

The persistent volume (PV) uses a Fibre channel connection to a 50GB LUN on a remote SAN. The access mode used was RWO (ReadWriteOnce). The QM is of type Single Instance (SIQM).

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 will be included in Appendix B. In the previous OCP internal report, three changes were made to the base MQ image to improve performance. These can now be applied by the MQ operator using supplied mqsc files during QM creation and can also be found in the same appendix.

The default cluster SDN (Software Defined Network) utilizes 1GbE networking. All master and worker nodes are also connected by an additional 10GbE 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.

The number of threads supported in a container in this environment is currently limited to 1024. To be able to run with more threads, a configuration change to the crio.conf file is required. cri-o is an implementation of the Container Runtime Interface (CRI) that supports Open Container Initiative (OCI) compatible runtimes. Again, please see the separate guidance on OpenShift configuration on how this was setup.

The first two sections in this report use a CPU limit of 32 cores for the QM. The CPU limit for the client is set to a value to avoid the client encountering CPU starvation.

Please see Appendix A for the specification of the hardware used for the cluster nodes.

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 64 primary files (of 16384 4K Pages) resulting in 1GB log allocation
- 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.

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 is 2 message puts and 2 message gets. The client runs within 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 not used as all messaging data flows over the private 10GbE network and the MQ client and QM are both running in the same OpenShift cluster.

Non Persistent Results

The graph below shows how the MQ QM performs for a 2K message size.

Figure 1 – 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; so 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 180,000 round trips/s, and even with just 16 requester client threads, the QM can achieve over 60,000 round trips/s.

This is an increase of over 3x when compared with the NP messaging performance in OCP 4.2

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

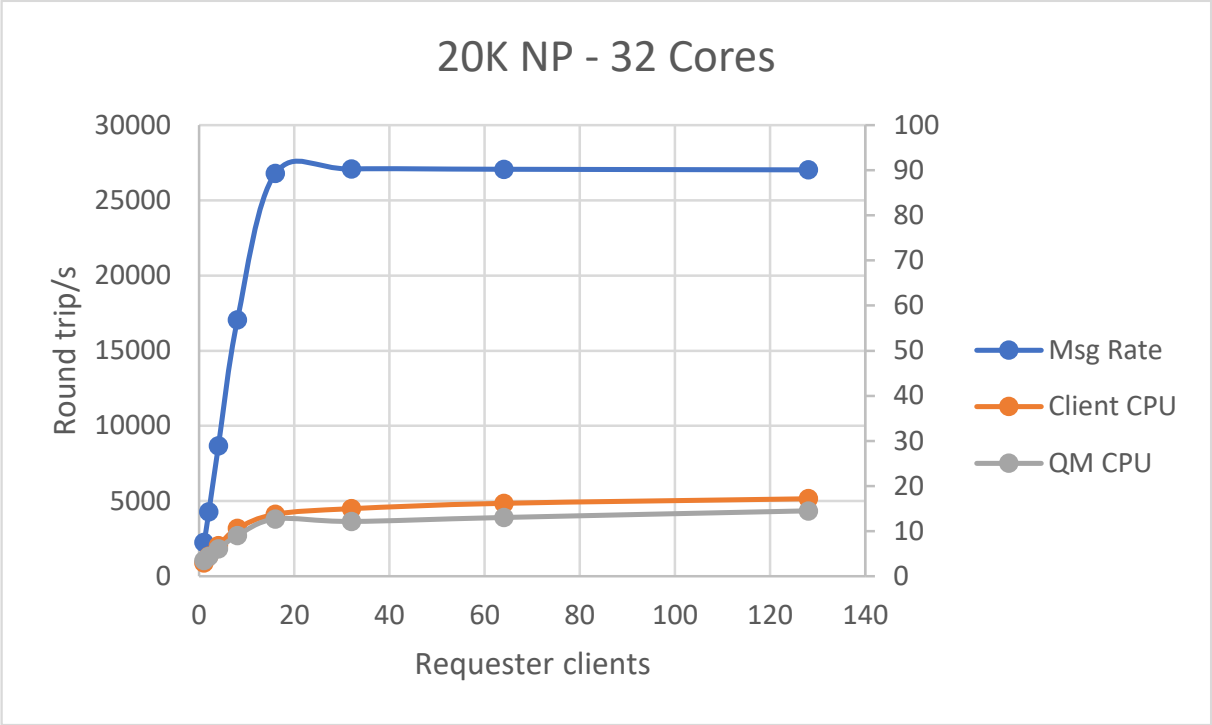

Figure 2 - 20K Non Persistent

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 network at over 27,000 round trips/sec, achievable with 16 or more requester clients.

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

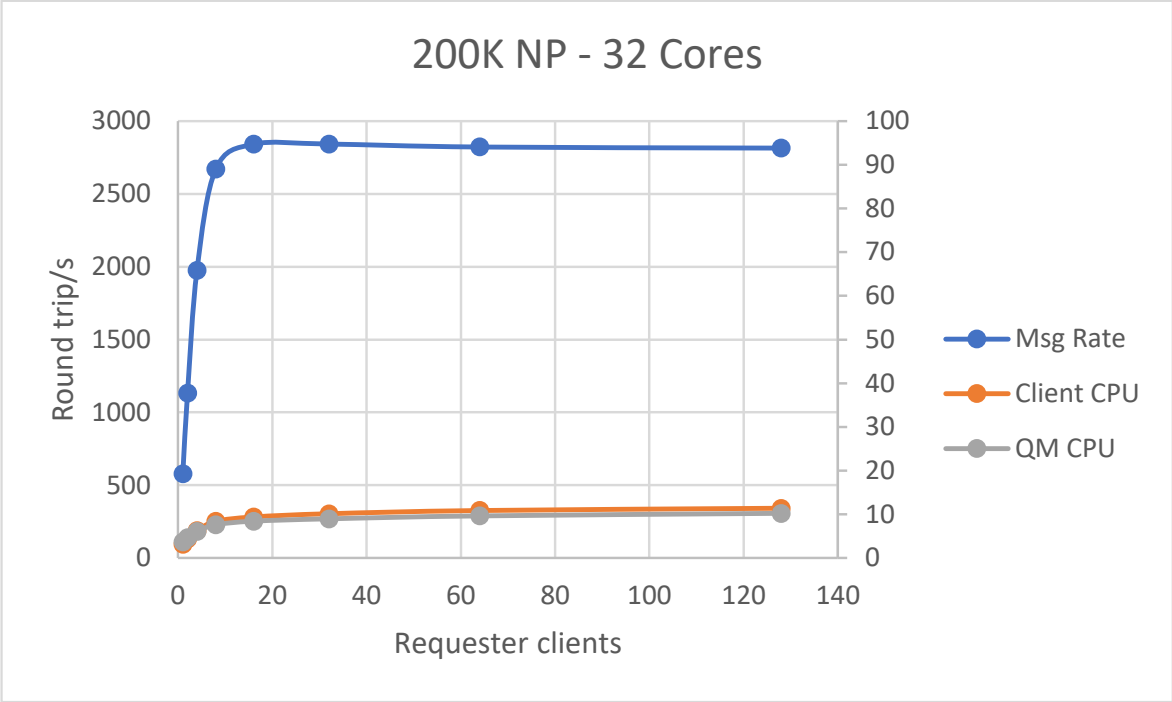

Figure 3 - 200K Non Persistent

The above graph again shows that we are limited by the network when the throughput has reached over 2,700 round trips/s from 8 threads and the QM CPU is less than 10% utilised.

Persistent Results

The graph below shows how the MQ QM performs for a 2K message size.

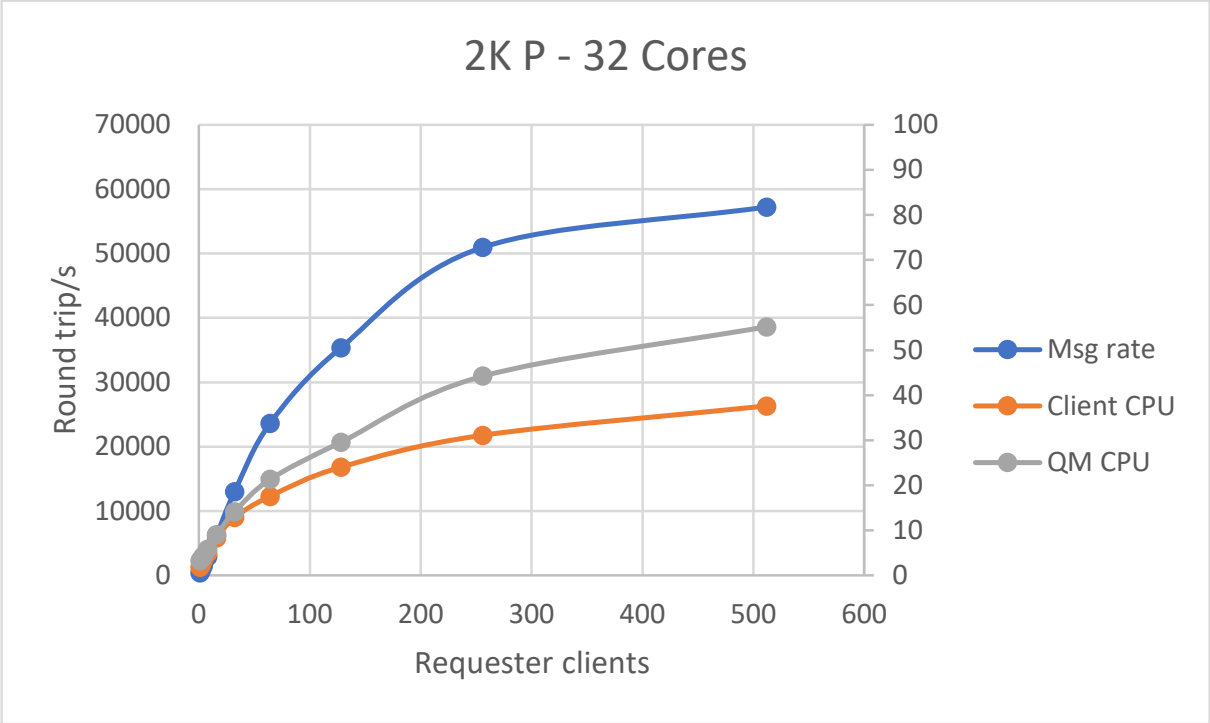

Figure 4 - 2K Persistent

The above graph shows that the QM can achieve a peak throughput of over 57,000 round trips/s until we saturate all the CPU available to the QM which has a CPU limit of 32 cores. This is over 60% faster than OCP 4.2.

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

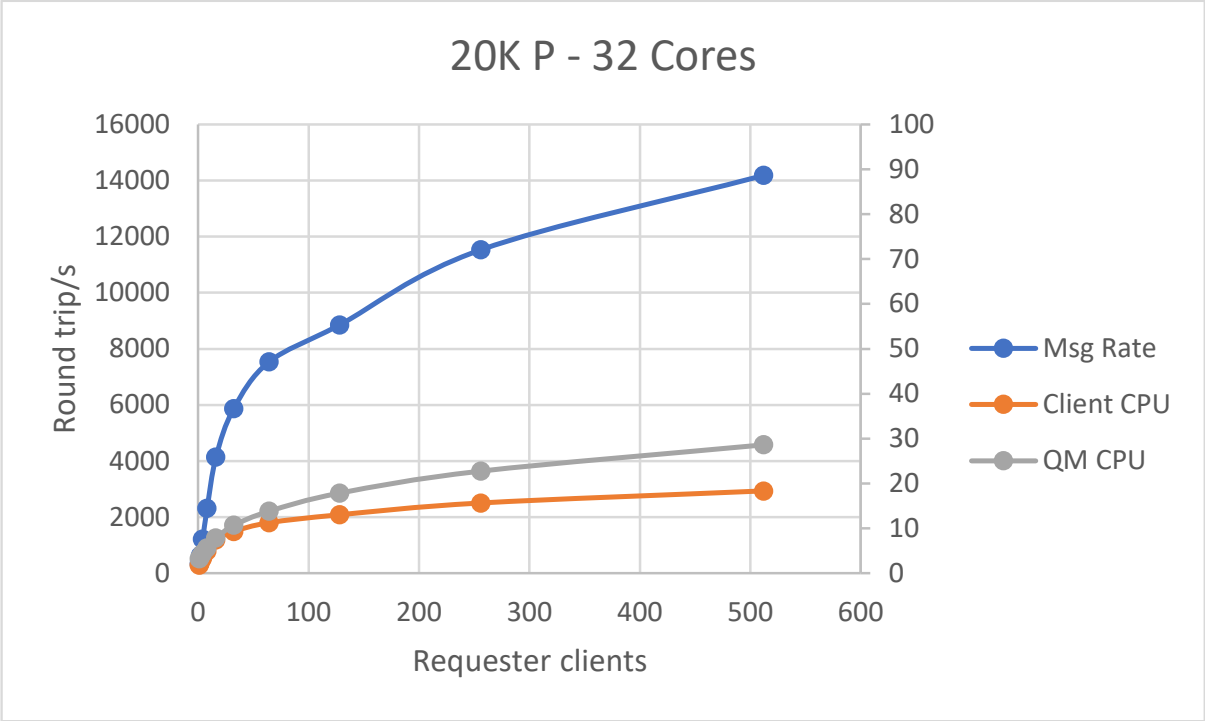

Figure 5 - 20K Persistent

The above graph shows that as we increase the message size to 20K, the QM CPU is no longer the limiting factor, and we can now achieve over 14,000 round trips/sec at 512 requester clients. This is over 60% faster than the same test in OCP 4.2.

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

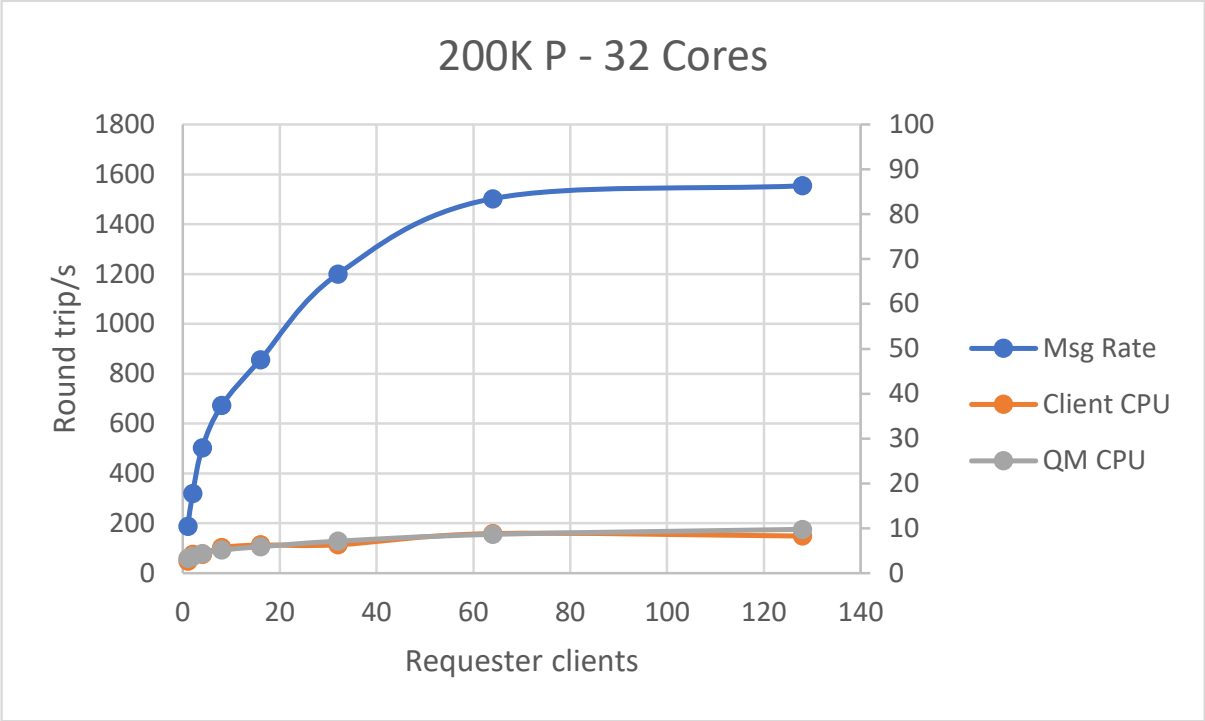

Figure 6 - 200K Persistent

The above graph again shows that we are limited by the persistence layer when the throughput has reached over 1,500 round trips/s and the QM CPU is less than 10% utilised. This is nearly twice as fast as the performance of the same test in OCP 4.2

Scaling Results

The results presented so far have been with a CPU limit of 32 cores, which is greater than we expect the majority of scenarios to use. 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 graph below shows how the MQ QM scales across varying CPU cores with a 2K message size.

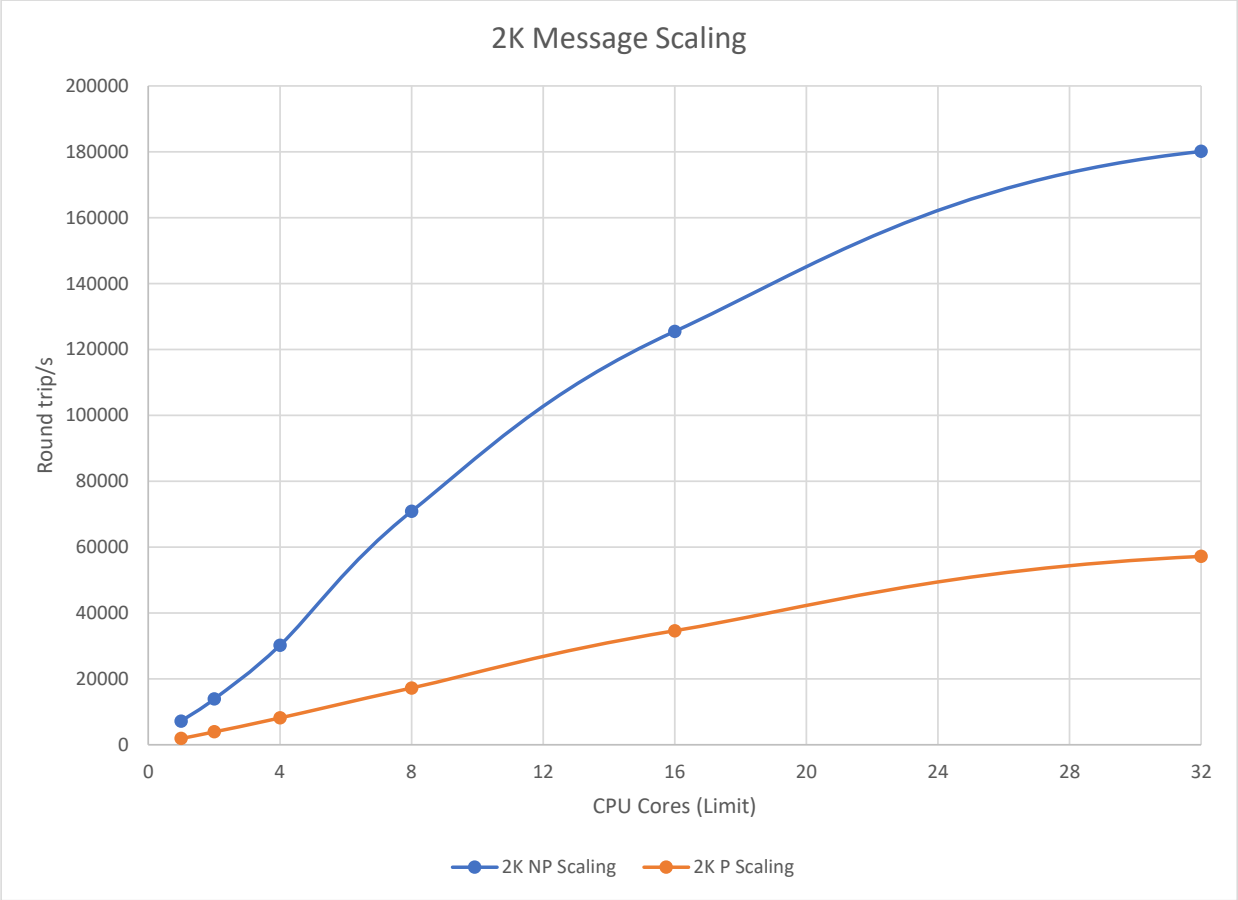

Figure 7 - 2K Scaling

The chart above shows how we can support over 7,000 round trips/s at a single CPU core right up to over 180,000 round trips/s at 32 CPU cores for Non Persistent messaging. For Persistent messaging the respective values are approximately 2,000 and 57,000 round trips/s.

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

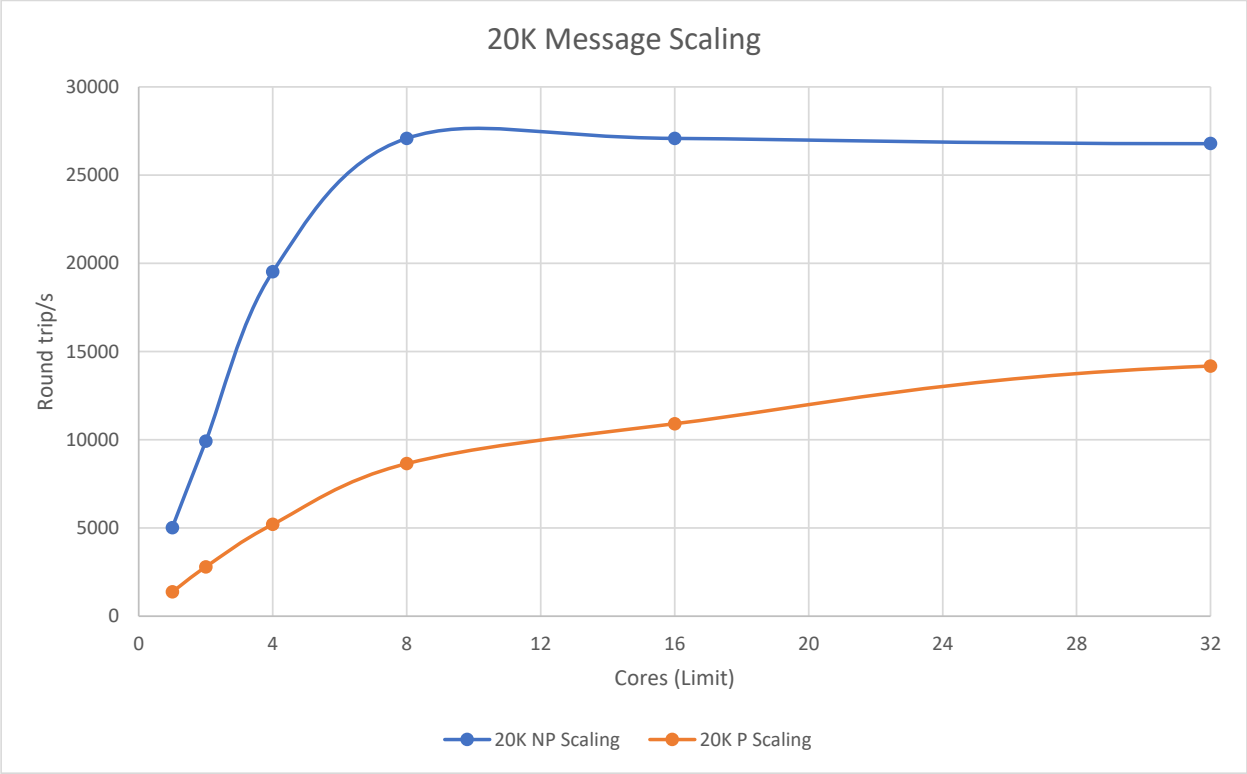

Figure 8 - 20K Scaling

The chart above shows how we can support 5,000 round trips/s at a single CPU core right up to over 25,000 round trips/s at 32 CPU cores for Non Persistent messaging, where the scenario has already become network limited. For Persistent messaging the respective values are approximately 1,300 and 14,000 round trips/s.

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

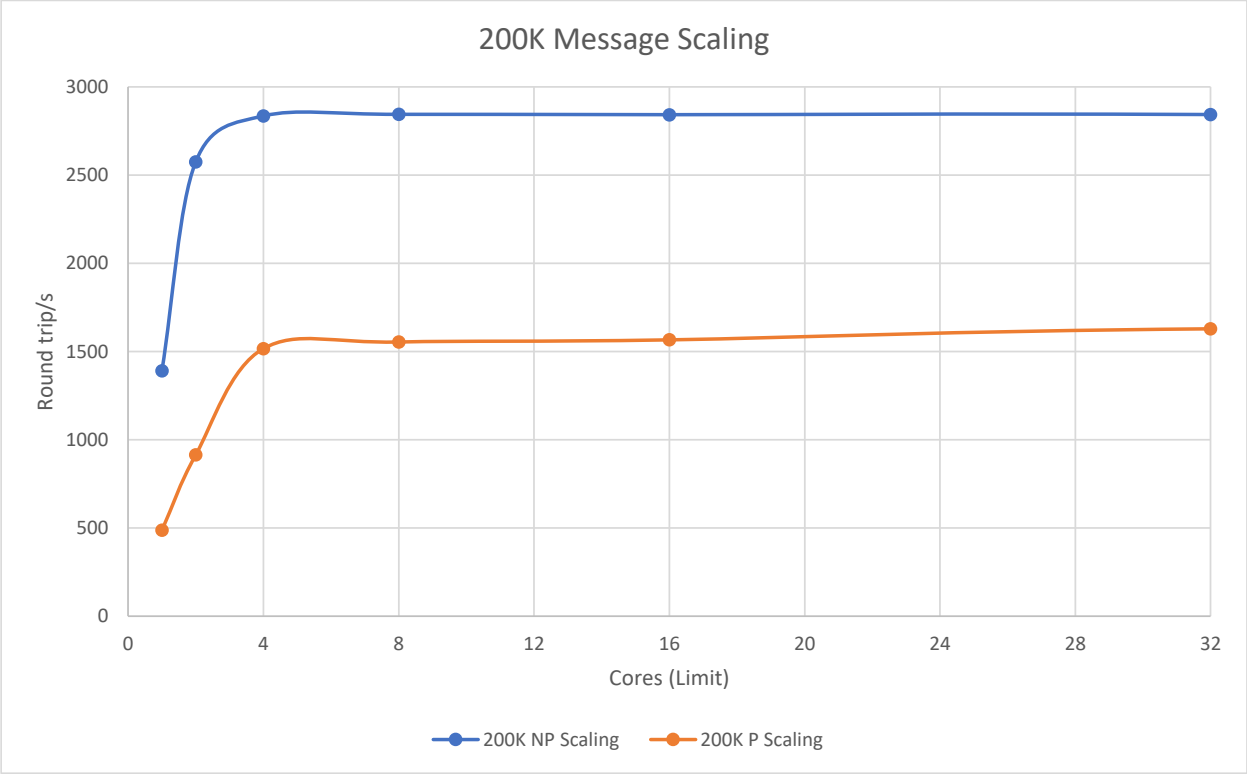

Figure 9 - 200K Scaling

The chart above shows how we can support 1,300 round trips/s at a single CPU core right up to 2,800 round trips/s at 4 CPU cores for Non Persistent messaging. For Persistent messaging the respective values are approximately 500 and 1,500 round trips/s.

Conclusions

In this whitepaper we have looked at the performance of the MQ QM in the OpenShift environment and shown the effect of varying message size, requester clients and CPU cores have on the performance of the QM.

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

Appendix A

Hardware specification for Worker Nodes:

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
spec:
  version: 9.3.0.0-r3
  license:
    accept: true
    license: L-RJON-CD3JKX
    use: Production
  web:
    enabled: false
  queueManager:
    name: "PERF0"
    availability:
      type: SingleInstance
    storage:
      defaultClass: san
      queueManager:
        enabled: true
    mqsc:
    - configMap:
        name: perf-mqsc-ini
        items:
        - disable.mqsc
        - queues.mqsc
    ini:
    - configMap:
        name: perf-mqsc-ini
        items:
        - fastpath.ini
    resources:
      limits:
        cpu: '32'
        memory: 16Gi
      requests:
        cpu: '32'
        memory: 16Gi
  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)
  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=64
LogSecondaryFiles=2
LogBufferPages=4096
TuningParameters:
DefaultPQBufferSize=10485760
DefaultQBufferSize=10485760
```