Queue Manager Restart Times for IBM MQ V9.1 & V9.1.1

March 2019

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

Introduction

This study will look at how long a queue manager takes to restart, in particular, when there has been a failure of some type, causing the queue manager to abruptly end, with potentially in-flight transactions. Some function associated with restart has been improved in IBM MQ V9.1.1, these are illustrated below.

A queue manager will maintain data, and transactional integrity, by synchronously logging operations to the recovery log. The logger component of the queue manager forces a synchronous log write every time a transaction is committed, either explicitly (through the use of syncpoints) or implicitly (e.g. an MQPUT outside of syncpoint).

Restart is typically quick, but as we shall see, there are some best practises which should be followed to avoid unnecessary delays (detailed in this blog article: *How long will it take to (re)start my queue manager?*). Tests presented here illustrate some of the points outlined in the blog article, with empirical data.

To recap, some significant impactors of queue manager restart time are:

- Amount of data on the log
- Active' Queue depths
- Checkpoint Frequency
- Queuing Style
- Long running or 'large' transactions
- Disk performance

We'll take a look at processing the recovery log, when the amount of data on the log, the queue depths, the number of long running transactions and disk performance is modified by the test. Checkpoint frequency and queueing style are not covered here.

Please note that many of the tests are not typical MQ restart times, some of the tests below are contrived to show how bad practises and environmental factors can affect restart time.

Restart Times for (Busy) Steady State Queue Manager.

These initial tests show restart times for a queue manager against which a request/response workload is running at a rate of 2,500 Requests/responses per second.

The workload is MQ-CPH based, using 2KB persistent messages (see the MQ-CPH blog article for a general description of the tool : https://www.ibm.com/developerworks/community/blogs/messaging/entry/MQ_C_Performance_Harness_Released_on_GitHub?lang=en_us)

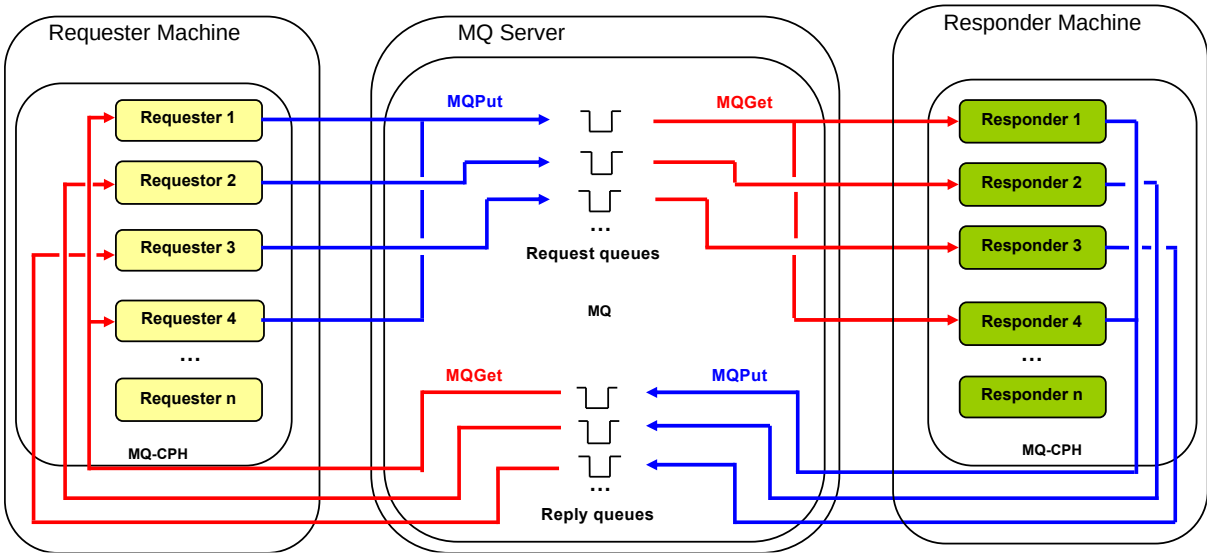


Figure 1 : MQ-CPH Requester/Responder Workload

Figure 1 shows the topology of the requester/responder test. The test simulates multiple 'requester' applications which all put messages onto a set of ten request queues. Each requester is a thread running in an MQI (MQ-CPH) application. The threads utilise the requester queues in a round robin fashion, ensuring even distribution of traffic.

Another set of 'responder' applications retrieve the message from the request queue and put a reply of the same length onto a set of ten reply queues. Each responder running in an MQI (MQ-CPH) application.

The test was run with 300 requesters, and 300 responders connected to MQ. The total rate of requests/responses through MQ was restricted to 2,500/sec and when the workload settled, the queue manager was killed [1] Start-up times measured are extracted from the queue manager's error log, and taken as the time elapsed between the following two messages:

```
AMQ5051I: The queue manager task 'LOGGER-IO' has started.
AMQ8024I: IBM MQ channel initiator started.
```

[1] A kill -9 command was issued against QM processes : sudo ps -ef | grep -i "amq\|runmq" | awk '{print $2}' | xargs -i kill -9 {}

AMQ5051I is the first message issued by the queue manager during start-up, and AMQ8024I marks the point at which applications can connect back in to the queue manager.

Prior to restart of the queue manager, the file system I/O cache is cleared on the MQ host machine, to emulate the state after a machine failure, or MQ being started on a different host (e.g. using MIQM).

```
sync;echo 1 > /proc/sys/vm/drop_caches
```

Note that this can make a significant difference to the restart time. As with all performance sensitive tests, you should run your own tests where possible, to simulate your production environment and circumstances you are catering for.

Results

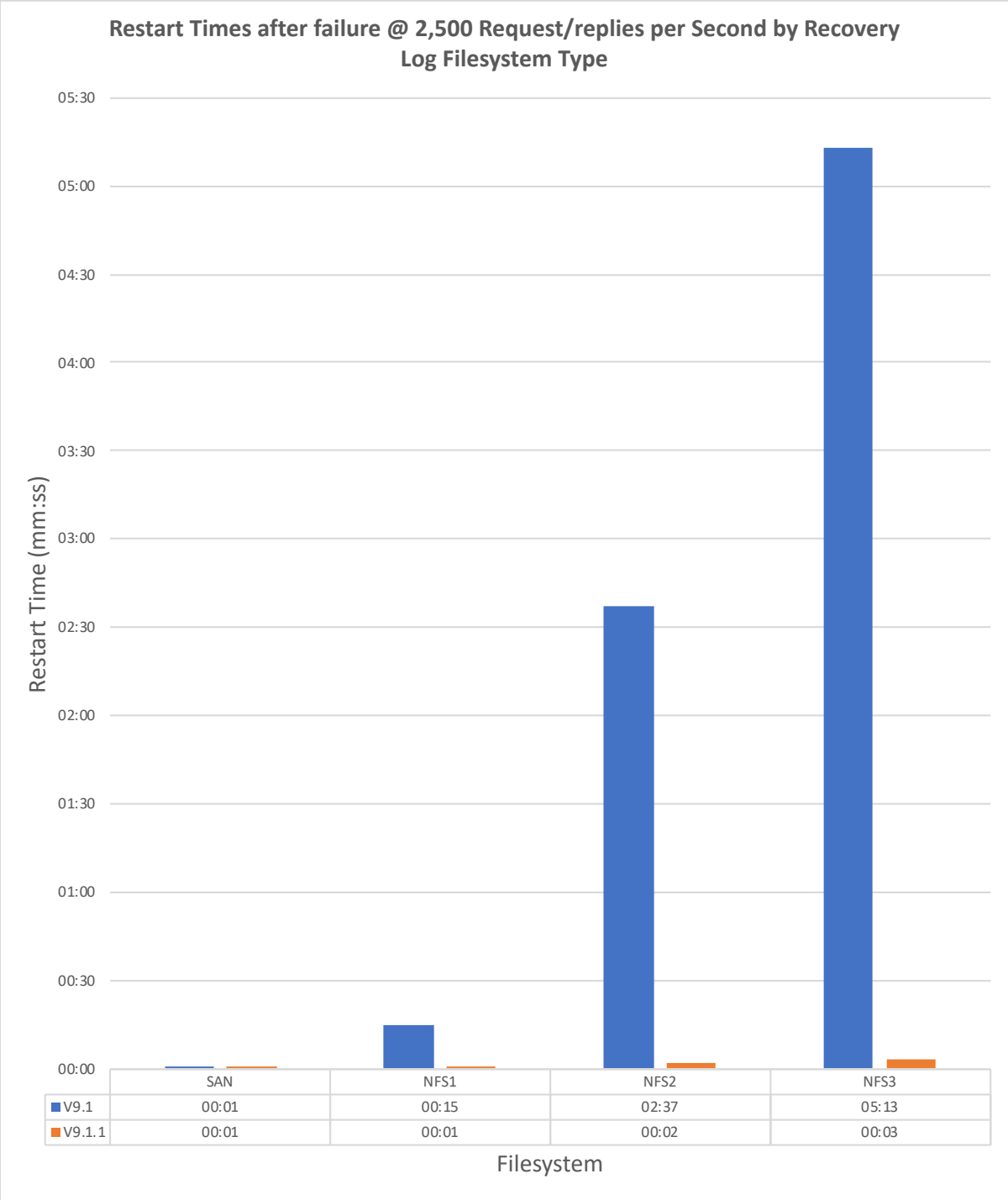


Figure 2: Restart Times for Busy QM

Figure 2 shows the restart times for the queue manager across a number of different file systems hosting the queue files and recovery log. The SAN file system has the lowest latency. The NFS filesystem is connected to via a 10Gb link, and NFS2 & NFS3 have additional delays added into the network – see Appendix 1 for more detail.

Restart times with the SAN hosted files are fast. When the higher latency NFS file systems are used, the restart times increase, with the bulk of the time being spent in the log record replay phase.

The number of log records replayed will depend on where the QM is logically, in relation to the last checkpoint. If a checkpoint has been taken very recently, there will be a small number of log records to be replayed, if the queue manager fails *during* checkpoint processing then there will be a much larger number of log records to be replayed. For the measurements above, several runs were measured. A linear relationship was exhibited between the number of log records replayed, and the restart time. The times above are normalised for 60,000 log records being re-played (NFS2 measurements, for instance, varied from 58 seconds to 4 minutes and 25 seconds in V9.1.0, depending on when the queue manager was killed).

Optimisations to reduce the file system interactions during log record replay have been introduced to V9.1.1, and restart times are dramatically reduced as a result. You should remember that these reductions particularly apply when (a) the file system's latency is high (NFS3 has a 1ms additional delay per file system interaction added), and (b) the file I/O requests are not met by the local O/S file system cache.

APAR IT27226 has been opened to make this optimisation available for V9.1.0 of MQ.

Impact of Deep Queues and Long Running Transactions on QM Restart Times

The previous section showed some restart times for a queue manager that was busy, serving a 'well behaved' application.

In this section I will run tests where there are deep queues to be recovered, or long running transactions causing the recovery log to fill, which will have an additional impact on restart time.

Remember that these are not typical restart times for IBM MQ, they are shown here to illustrate what to avoid.

For all of the tests below I used a simple 'putter' application adding messages at a rate of 5000 per second to 10 queues and killed the queue manager whilst there were one or more outstanding transactions (MQPUTs) for each queue.

Note that if populated queues exist that were not involved in transactions when the queue manager was killed, then the messages on those queues will not be loaded until they are referenced by an application, after start-up. For the tests below, all of the deep queues are loaded as part of the queue manager start-up, as there are transactions in-flight on all of the queues being used in the test.

The recovery log file parameters used (unless otherwise specified), were:

LogFilePages=16384
LogPrimaryFiles=32
LogSecondaryFiles=2
LogType=CIRCULAR
LogWriteIntegrity=TripleWrite

This results in 32 x 64MB primary log files, providing a total of 2GB of primary log space.

All of the tests, unless otherwise specified, utilised a SAN based file system attached to via an 8Gb fibre link, and utilised a SAN volume controller (SVC).

Amount of Data in the Log and Long Running Transactions

Let's take a look at what happens when we give MQ some work to do on restart, by filling up the recovery log with data.

The recovery log files are processed as part of the queue manager start-up. This takes place in two main phases:

1. Replay the log (read all data from the active recovery log and update the queue files with messages that were logged after the last recorded checkpoint). This is

sometimes referred to as the REDO phase of a transaction manager's restart protocol.

2. Backout any uncommitted transactions (removing the associated messages from the queue), this is sometimes referred to as the UNDO phase of a transaction manager's restart protocol.

Queues are loaded into memory as required during phase 2 (that is the queue's meta-data, rather than the actual messages themselves).

Figure 3 below shows the impact on start-up time, as the log becomes filled with committed, or uncommitted messages. The tests all start by putting a single 2KB message on each of our 10 (empty) test queues (without committing them), then adding additional 2KB messages, which fills the recovery log, as the initial messages are stopping MQ from freeing up log space, even when the additional messages may be committed.

The test was run with the following numbers of additional messages:

# Messages	Primary Log Used
10,000	~5%
100,000	~23%
400,000	~80%

Beyond 80% of primary *and* secondary log usage, MQ will forcibly free up log space, by abending the initial transactions, and backing them out (application will receive error code 2003 - MQRC_BACKED_OUT).

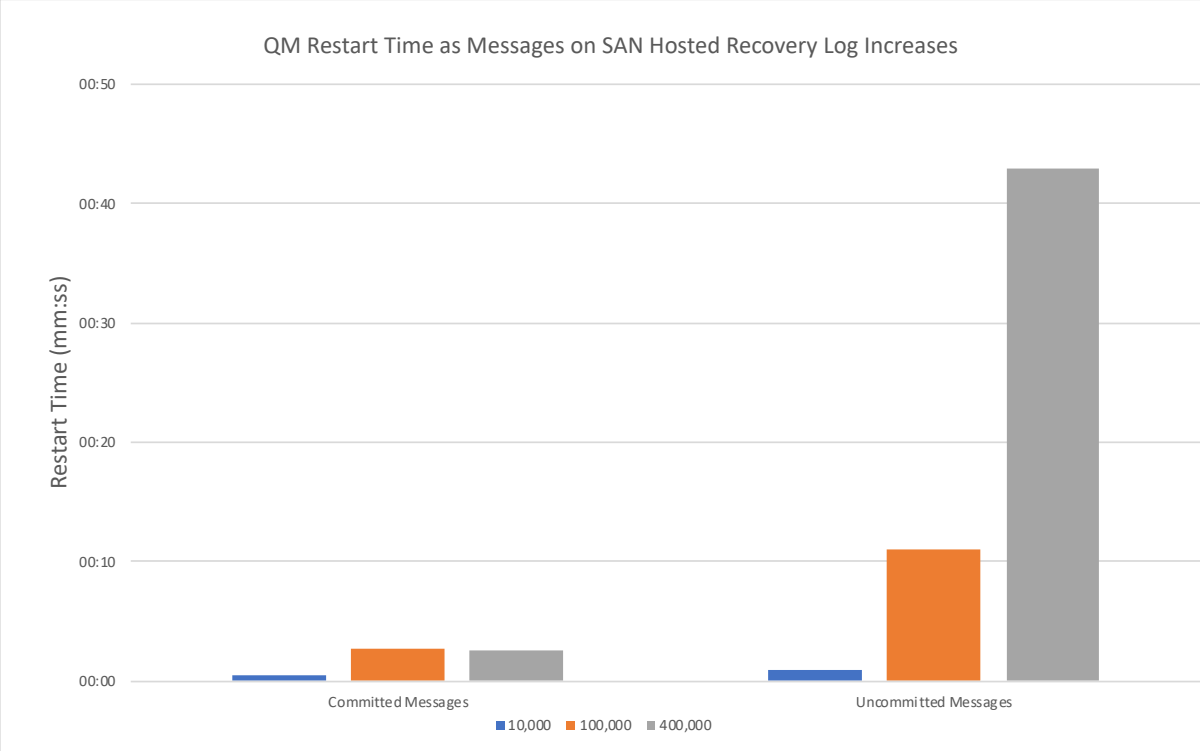


Figure 3: QM Restart-times for Heavily Populated Recovery Log (V9.1.0)

Results for IBM MQ V9.1.1 were very similar to V9.1.0 for these tests.

The amount of data read from the log files during phase 1 is the same whether it is comprised of committed, or uncommitted data. Phase 1 of the restart was about the same for both cases and constituted most of the restart time when the log files were full of committed messages.

Although log replay was not a big part of the restart time with SAN storage (used in the tests above), this phase can take much longer when the file system has a higher latency (demonstrated later in this report).

Phase 2 involved loading queues (1 message on each queue for the committed case, and up to 40,000 per queue in the uncommitted case), and recovery. This phase was insignificant for the committed messages test but constituted the bulk of the time taken for the uncommitted test case.

The test case is somewhat contrived, to force the log to be filled to known states, by having the 10 initial long-lived messages, in both cases. Large, full log files are not good news in production. Long lived transactions will eventually fail when the log fills completely and if the rest of the log contains a lot of uncommitted data, that will cause delays in the event of recovery/restart.

So large logs, filled with uncommitted messages will delay a restart of the queue manager. There can be many transactions, or just a few, what really matters is the total number of uncommitted messages, as each of those has to be reconciled with the queue. The table below shows this. If the message size is increased, the same amount (in KB) of uncommitted application data takes less time to recover as there are less messages to reconcile. Spreading the number of uncommitted messages across 10, or 3,200 transactions makes little difference, but our case is exceptional, with all transactions being backed out in the restart. Having very large transactions should generally be avoided, as just a single transaction requiring a backout could represent a lot of messages to be processed.

Msg Size	#Messages	#Transactions	%Primary Log Utilisation	Restart Time
2KB	400,000	10	78%	49 seconds
2KB	400,000	3,200	78%	46 seconds
20KB	40,000	10	51%*	8 seconds

*The lower log utilisation is due to the fact that meta data (message headers & internal meta data) is proportionally lower for larger messages. The total application (message body) data is 800MB in all cases.

Deep Queues

For the remaining tests we will look at the 10,000 committed messages case (with 1 uncommitted transaction on each queue, ensuring queue loading will take place as part of the recovery, thus enabling consistency of measurement).

If the queues involved in recovering transactions already contain a lot of messages, then the loading of these queues, as part of the recovery process, is going to take longer. In Figure 4 below, the test with 10,000 committed messages on the log, was re-run against queues pre-populated with messages (250,000, 500,000 and 1,000,000 on *each* queue).

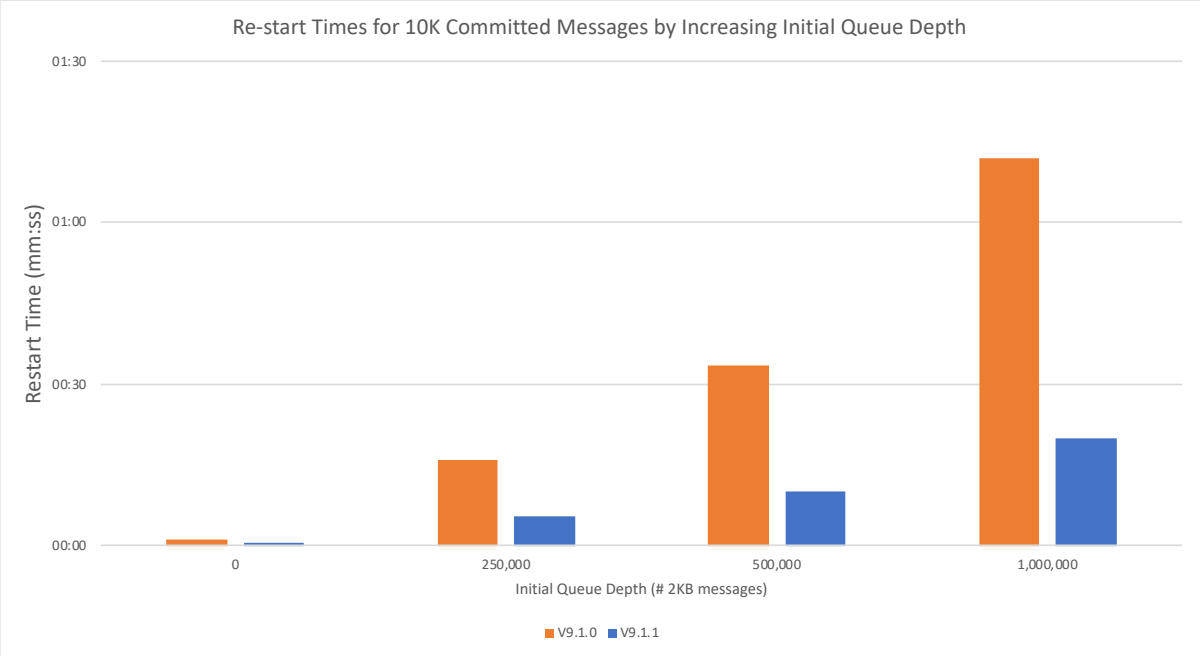


Figure 4 : Restart Times with Deep Queues, after 10K Committed Messages

With deep queues there is more data to load from disk during the recovery phase, and more messages to scan through, when backing out transactions. Deep queues should be avoided if possible, as they also involve more disk activity during normal operation, and any use of selectors (especially on message header fields) will take longer, due to the larger number of messages to scan though.

From the IBM MQ V9.1.1 release, queue loading of multiple queues, as part of the recovery process is executed in parallel, speeding up restart times. This can be seen in Figure 4 above.

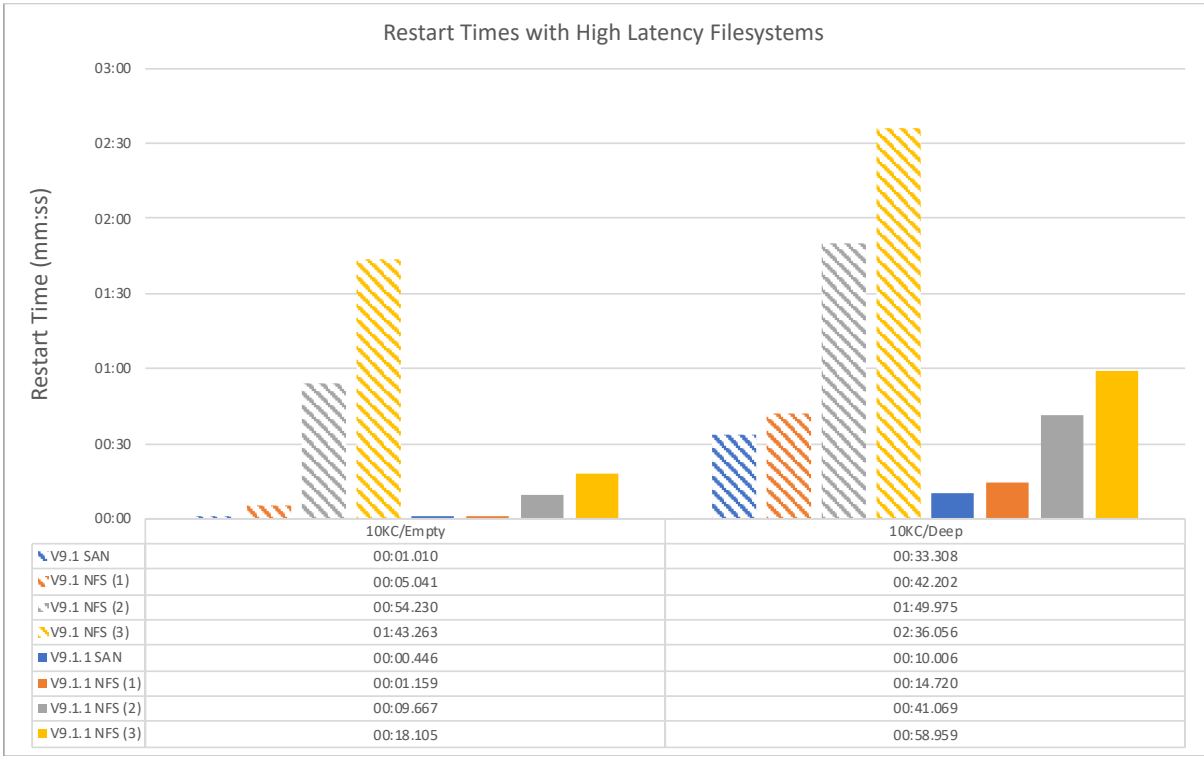
High Latency File Systems

Restart and recovery are I/O intensive, so if latency is introduced, it can have a significant impact on restart times. Figure 5 below shows two tests, previously presented, but re-run with the queue manager log and queue files hosted on NFS. For two of these tests, we introduce a network delay to increase the latency of the file systems, to demonstrate the impact on restart times. See Appendix 1 for more detail.

The tests are:

10KC/Empty 10,000 committed messages on the log, with previously unpopulated queues

10KC/Deep 10,000 committed messages on the log, with pre-populated queues (500,000 messages on each queue).



	10KC/Empty	10KC/Deep
V9.1 SAN	00:01.010	00:33.308
V9.1 NFS (1)	00:05.041	00:42.202
V9.1 NFS (2)	00:54.230	01:49.975
V9.1 NFS (3)	01:43.263	02:36.056
V9.1.1 SAN	00:00.446	00:10.006
V9.1.1 NFS (1)	00:01.159	00:14.720
V9.1.1 NFS (2)	00:09.667	00:41.069
V9.1.1 NFS (3)	00:18.105	00:58.959

Figure 5 : Restart Times for High Latency File Systems

Results for both V9.1 and V9.1.1 are shown, as these tests benefit from optimisations to queue file access during restart, introduced in V9.1.1 (APAR IT27226 on V9.1).

Higher latency file systems shift the primary overhead of start-up from reconciling transaction data, to file I/O during the log replay phase, so the optimisation of this phase in V9.1.1 particularly helps with higher latency filesystem, reducing the amount of calls across the slower link.

Parallel queue loading helps as well, but as the latency of the filesystem increases the critical factor is the reduction of I/O calls rather than whether the queues are loaded in parallel, or not.

Restart times for high latency file systems can be even longer if there are a lot of uncommitted messages on the queue. We have seen this has a big impact when tested on SAN hosted recovery logs. If this extreme case were to occur on a high-latency file system, the recovery phase could take tens of minutes to complete.

Conclusions

As we have seen, the time it takes to restart a queue manager is largely dependent on the time it takes to replay the recovery log records and recover in-flight transactions. This in turn is impacted by the amount (and type) of data on the log, whether a checkpoint has recently been taken, and the quality of the file system hosting the recovery log. Some aspects are largely outside of our control (we can increase the frequency of the checkpoints for instance, but the point at which a failure occurs in relation to when the last checkpoint was taken is out of our control).

Remember,

- Large numbers of uncommitted messages should be avoided where possible.
- Deep queues will slow down restart times, especially where transactions are being backed out on those queues.
- Parallel queue loading, during recovery (introduced in V9.1.1) can dramatically reduce restart times, especially for faster filesystems.
- Recovery of in-flight transactions is much improved in V9.1.1 , and this can particularly help the restart times for queue managers whose recovery logs are hosted on higher latency file systems.
- Recovery logs should be sized for well-behaved applications. Catering for long running transactions can contribute to a much heavier overhead during restart, recovering the state of the queue manager.

Following best practises, along with improvements made in MQ V9.1.1 should ensure speedy queue manager restart times but testing your own environment will enable you to know what to expect.

- Test your recovery scenarios, making sure the environment matches that in production as closely as possible.
- Understand the performance of the file system hosting the recovery log, and the network hosting an NFS recovery log.
- Record expected re-start times for different load scenarios, along with any other metrics that might be useful to compare to, should a queue manager failure occur in production
- Remember that some parts of the restart can be highly variable (e.g. whether the queue manager is re-starting on the same machine or not, and if so, whether OS file caches are intact, or not). Test for the most likely scenarios.

Appendix 1. File systems

For all of the tests run in this report, the queue files, and recovery logs were hosted on one of the four file systems listed below.

SAN: SAN hosted file system, using an IBM San Volume Controller (SVC), via 8Gb fibre links.

NFS1: NFS (V4) hosted on RAID cached disks via a dedicated* 10Gb network link.

NFS2: NFS (V4) hosted on RAID cached disks via a dedicated* 10Gb network link with an additional 500us delay on network added in each direction.

NFS3: NFS (V4) hosted on RAID cached disks via a dedicated* 10Gb network link with an additional 1ms delay on network added in each direction.

*All test machines were connected to a local, switch via 10Gb links, where there was no competing traffic on the switch.

Network delays were configured using the Linux traffic control (tc) utility. E.g. for NFS2 the following commands were executed on the NFS server and the MQ host.

```
intf=<10Gb interface>
delay=500
tc qdisc add dev $intf root netem delay ${delay}us
```