Implicit syncpointing for persistent messages put outside of syncpoint

In MQ v9.0.5 implicit syncpoint enables puts of persistent messages outside of syncpoint to perform better when there is contention for the queue. This happens when there are multiple applications putting to the same queue.

If there are messages you can afford to lose, make them non-persistent and put and get them without syncpoint. For messages that you do not want to lose, make them persistent and put and get them inside syncpoint. MQ is optimised so that putting persistent messages inside syncpoint generally performs better than putting them without syncpoint. This is because the queue is locked for a shorter time when putting inside syncpoint, so there will be less contention for the queue when multiple applications are putting to the same queue.

In v9.0.5, the queue manager relieves the contention problem of puts outside of syncpoint by optionally adding an implicit syncpoint to persistent puts of messages outside of syncpoint. So when your application puts a persistent message without syncpoint, the queue manager will actually put it under syncpoint and then commit the implicit transaction before the MQPUT returns.

This is controlled using a new tuning parameter ImplSyncOpenOutput which you put in qm.ini. Here's what I appended to the qm.ini of my queue manager….

```
TuningParameters:
ImplSyncOpenOutput = 4
```

This means that the queue manager will add an implicit syncpoint to a put when 4 applications have the queue open for output. By default, ImplSyncOpenOutput is 2 if you don't set it in qm.ini. 2 is a good default because that means that implicit syncpoint will always be used unless no other application has the queue open for output. If there is only one application putting to the queue, implicit syncpoint may go slightly slower because it requires 2 log writes instead of 1. And if there is only one putter, there is no contention so there is no advantage in adding an implicit syncpoint. So that's why ImplSyncOpenOutput defaults to 2 and not 1.

If you always want implicit syncpoint, set ImplSyncOpenOutput to 1. However there are some occasions when the queue manager doesn't add an implicit syncpoint. One such occasion is when the application already has an active transaction, when interleaving putting persistent messages with and without syncpoint. But if you want to switch off implicit syncpoint completely, set ImplSyncOpenOutput = OFF.

If you're happy with the default value if ImplSyncOpenOutput, there's no need to set it in qm.ini, in which case you'll get an implicit syncpoint whenever multiple applications are putting to the same queue. So you'll enjoy the improved performance by doing nothing except upgrading to v9.0.5.

Performance data to show the benefit of implicit syncpoint

Figure 1 shows results for tests where 10 queues pairs are utilised, with an increasing number of requester applications running, processing 2KiB messages. When there is only one application, there will never be another MQPUT being processed alongside that of application 1, so there is little difference between executing the MQPUT inside, or outside of syncpoint, in the application. Once we add more MQ applications, the benefits of using syncpoints become evident, particularly with a higher latency filesystem, as MQPUTs outside of syncpoint will lock the queue while the log record is synchronously forced to disk. Using syncpoints reduces contention with the added benefit that other applications can write into the log buffer, resulting in more aggregation of log data, in a single write. With V9.0.5, implicit syncpointing reduces lock contention, matching the performance of the explicit syncpoint scenario.

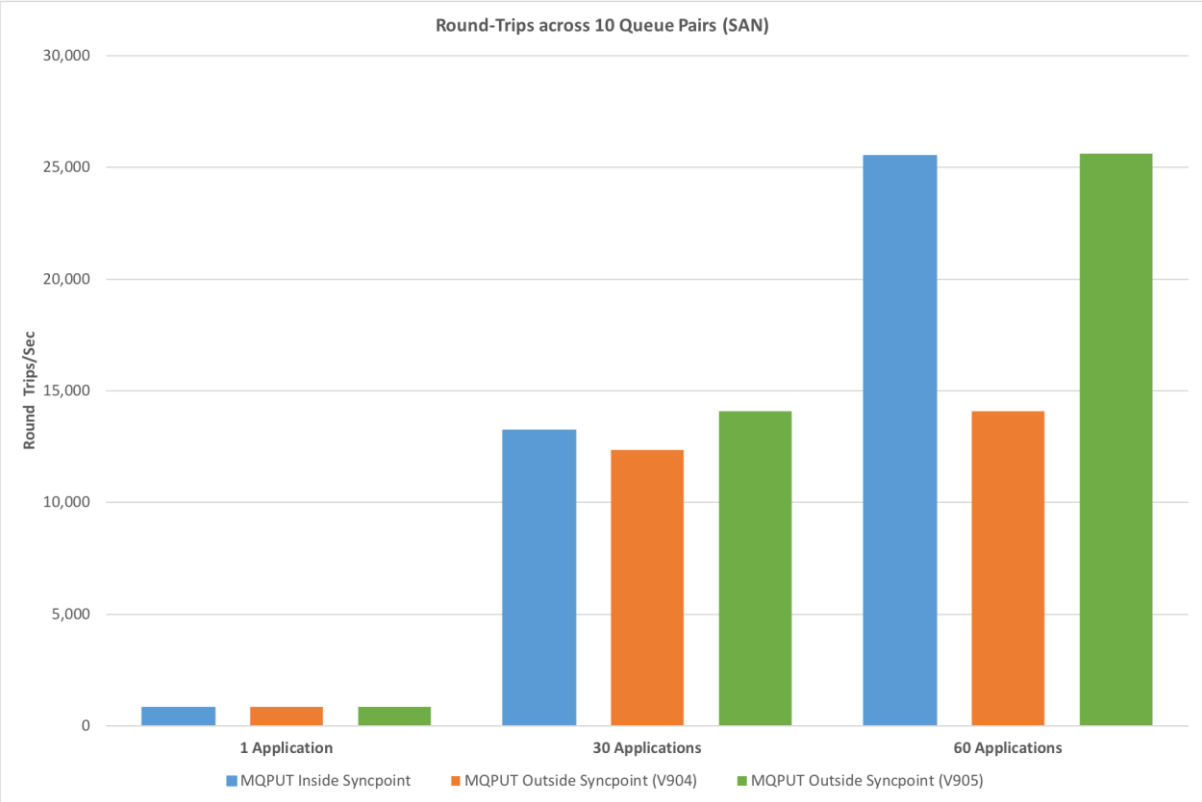

Figure 1

Figure 2 shows the effect of reducing queue locking by spreading the load across a number of queue pairs (REQUEST Q/REPLY Q). All tests use 60 requester applications. When the workload is driven through a single pair of queues, the non-syncpoint case has a low throughput (not much better than the test using 1 requester in chart 1), as each MQPUT queues up behind the previous one to that queue, with a forced log write being executed within the scope of the queue lock. Using syncpoints alleviates this issue, allowing for more concurrency. As we increase the number of queue pairs, the locking becomes less of an issue, until, at 60 pairs of queues, where there are only 2 requester applications per queue pair, the non-syncpoint case is not much less than using syncpoints. Once again, the V.9.0.5 test case, with PUTs outside of syncpoints matches the explicit syncpoint scenario.

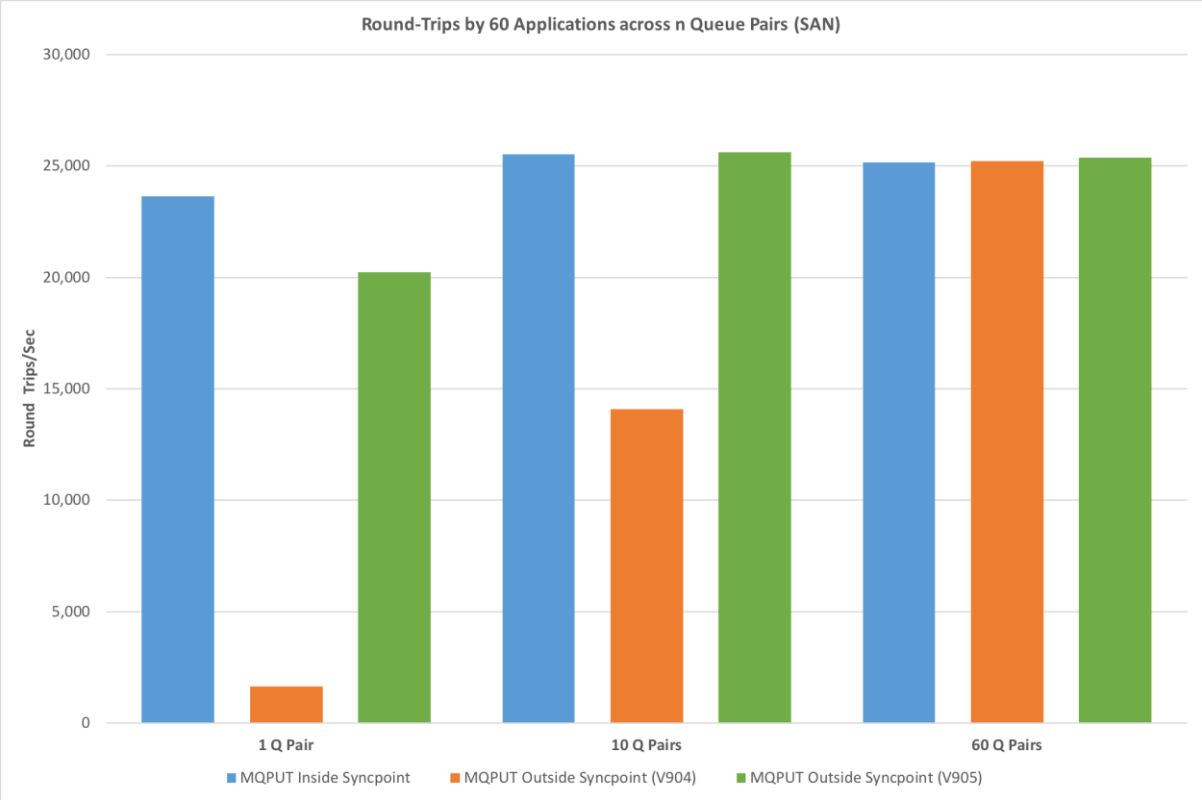

Figure 2

The test case run was a round-trip scenario, where a putter application puts a message onto Q1, a getter application gets the message from Q1 and puts a reply message onto Q2, and then the original putter application retrieves the reply. This test was run with the initial PUT of the message by the putter application being inside syncpoint (run on V9.0.5, but the performance of this test is similar to its equivalent on V9.0.4), and outside of syncpoint (run for V9.0.4 & V9.0.5 with the new implicit syncpoint functionality). All tests have the MQ transaction log hosted on a SAN mount point.

Round trip case 1 (requester application PUTs inside syncpoint)
Requester Application(s)
PUT (inside syncpoint) -> Q1
GET (inside syncpoint) <- Q2
Responder Application(s)
GET/PUT (inside syncpoint) <- Q1 -> Q2
Round trip case 2 (requester application PUTs outside of syncpoint)
Requester Application(s)
PUT (outside of syncpoint) -> Q1
GET (inside syncpoint) <- Q2
Responder Application(s)
GET/PUT (inside syncpoint) <- Q1 -> Q2