Shared Queues and Notifications – Oh Yes It Depends!

The Coupling Facility will notify all queue managers attached to a Queue Sharing Group when a queue goes from empty to non-empty. We have long focused on over triggering, because that has traditionally been more of a sore spot for several reasons. Today I am going to describe 'over notification.'

Over notification can occur when there are multiple instances of an application in a get-wait state and the messages are arriving at a slower rate than they can be processed. For example, I have 8 queue managers each with two instances of a CICS transaction that are in a Get Wait state – that is there were no messages for this instance to process so the application waits for the next notification. When messages are coming in relatively slowly, this will mean that the arrival of a single message on the shared queue will result in as many as 16 transaction instances getting notified. All 16 will resend the MQGET and whichever queue manager gets to the CF hosting the queue first will get the message and all the other instances will get 2033 and potentially go back to a wait state or will end if their wait interval has expired. This, of course, leads to a lot of unfulfilled MQGETs and the overhead of notification, etc.

Is this a problem? It depends! If there are 16 instances is because multiple times per day there are multitudes of messages unleashed due to regular workload processes, then over notification or over allocation of application instances, may be necessary to get the workload completed in a timely fashion. If this is not the processing pattern for this queue, or if the rest of the business day is processing a very limited number of messages, then there can be substantial overhead because of the over notification.

So how do we know if this is happening, or if it is a possibility? Some of the clues are the same as for a polling application. A polling application is one where the application (often a client application) is not using gets with a wait but using a timing mechanism to issue an MQGET to see if there is work available. The first clue is comparing the number of MQGETs reported by the Message Manager and the number of gets reported by the Data Manager. If a queue is empty, the message manager will not send a request to the Data Manager, those 'empty gets' show up. This clue alone does not tell us if there is a problem or what the problem may be but tells us we must look deeper.

With MQ 9.4 and above, this issue may be a bit easier to identify as the Queue Statistics report on more application behavior than other statistics, including a count of unfulfilled or empty gets, count of get-waits being re-awakened, etc.

If you are on MQ 9.3, then you must look at the Task accounting data (Accounting Class 3 data), to see more of what is going on. The task accounting data includes the task Identification record (WTID), the task statistics data (WTAS), and task associated queue (WQ) records for each queue used as part of the task. Note – if ACCTQ is turned off for queues under evaluation, data will need to be collected again with ACCTQ on.

When we, the WSC, look at this data, we calculate the percentage of valid gets. If the queue is heavily used, and the get percentage is very low (less than 10-15%) across multiple collection intervals it is time to look a bit closer. If the valid get percentage is greater than 50% we typically don't worry about extra overhead. However, if it is consistently very low (0.00%-15%) then examining the reasons why can be worth it. If the queue is not heavily used, and the percentage is low, we often flag it as something to be watched.

Is this a case of application polling? At the WSC, we often do not know much about the customers applications and are finding that in some cases the MQ for z/OS administrators may not know much about the applications themselves – especially when they are client connected applications. If the application is long running and crosses SMF interval boundaries, examining the number of API requested MQGETs across multiple intervals will show that on average the number of API MQGET requests is the same or very similar values AND the task is associated with a SVRCONN channel, chances are good that the application is polling for data arrival. There are some Java application environments that default to polling rather than using a more efficient get-wait model, as that is easier to code. Polling for message arrival is usually inefficient on any platform, we just notice the inefficiency more on z/OS because of the way product is licensed.

If the application is not a client connected app, it could still be polling but based on our observations that is less likely. The same type of evaluation should be done for other connections (batch, CICS, etc.)

The application is not polling – how can I tell if I really have an over notification situation? One indication is in the Type 24 latching, as reported in the WTAS records. MQ, like most subsystems, must serialize some types of requests and 'lock' resources while they are being used. MQ uses 32 latch types, most have multiple uses. Type 24 latches are used to serialize across multiple waiting getter instances. What we, the WSC, observed is that when there are many application instances waiting and the messages are trickling in, the number and length of time these type 24 latches are seen are typically much higher than when the messages are coming in at a rate to keep all the instances processing, not just waiting.

In one particularly interesting situation a customer increased their queue managers in a QSG substantially to reduce the log task busy percentage to a level where more work can be absorbed. They were running at over 99% log task busy in their QSG queue managers during our review of this new government mandated process, which means that additional persistent workload would cause more waiting on logging. Increasing the number of queue managers processing the workload is the traditional advice for reducing the log task busy percentages.

Introducing new queue managers into the QSG to resolve the log task busy issue worked to lower the log task busy issue, but introduced a significant amount of additional overhead, with no real increase in message volume. We did not anticipate the substantial increase in CPU use, and this is in part due to not having examined the actual flow of messages and activity on the individual queues. Once we began examining the queue level data following the increase, we saw that some queues were getting triggered a lot more (one particular queue had gone from a valid get rate of about 3% (quite low) to less than half a percent during the hours of collection.

It also occurred to us that there may be a fairly simple way to address the issue. KEYRNOTIFYDELAY (KRND) has been identified as the method to reduce over triggering on shared queues. Basically, this attribute of a CF structure limits notification to a single queue manager when a queue transitions from an empty to a non-empty state. If there are still messages on the queue when the DELAY interval expires, then all the queue managers are notified. This helps avoid the problem seen with earlier implementations of throttling notifications, that is when the messages start arriving at a higher rate but new application instances are not started because they show as active. What we were less confident about was whether KRND would also work for waiting getter notification. After a discussion with our z/OS colleagues, we thought that the notification from the CF was likely to be the same for triggered queues and for waiting getters. The WSC does not have access to the MQ source code and this was not discussed as a possibility when KRND was first implemented, so we were not sure. What we had observed is that using this attribute did seem to help in our little test environment, so we suggested testing this to the customer. The results were astonishing.

Each of the queue managers had a reduction in the overall number of 5K microsecond waits for Type 24 latches from 4000+ to 150 (or less) for the two hour data collection. This was true even though for some queue managers the volume of MQPUTs to the queue was higher than the previous set of data. This, along with the application of KRND on a structure hosting triggered queues, reduced the overhead significantly, and reassured us that the zero-to-non-zero notification from the CF to the queue managers was the same no

matter what kind of notification (trigger or wakeup call) is needed to send to the application. We suspect that tuning the delay attribute may help even more, but that will depend on the workload and physical Parallel Sysplex set-up.

At the request of the customer, the development lab will be publishing an article on this their test findings.