Multiple cluster transmission queues – what, why and how

[Jamie Squibb](javascript:;) |Sep 20 2016 Updated

I have recently received a couple of queries from customers regarding the use of multiple cluster transmission queues with IBM MQ. One customer wanted to know how best to update an existing cluster to use them. The other customer had a cluster-sender channel that wouldn’t start because they’d accidentally deleted its transmission queue and wanted to know how best to recover. It is actually quite easy to resolve that problem and fortunately it was only on a test system so the customer wasn’t unduly impacted. Given these queries I thought a blog post on this subject might be useful. This post describes the multiple cluster transmission queue feature, why you might wish to configure multiple transmission queues and how to do so. I also explain how to resolve some problems you might encounter. There is quite a lot of information in this post so I’ve divided it in to sections so you can jump straight to content that interests you.

**Introducing multiple cluster transmission queues...**

A transmission queue is used by MQ to store messages until they can be transmitted over the network to their destination. Regular sender channels have a dedicated transmission queue and they send all messages put to their transmission queue to their remote receiver. Cluster-sender channels are more cooperative. The default behaviour is for all cluster-sender channels to share a single transmission queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlation ID of messages put to this queue identifies the cluster-sender channel over which they should be sent. Cluster-sender channels use MQGET by correlation ID to remove only those messages they should send. I mention this because many users don’t realise this difference compared to other transmission queues. It is actually extremely important that cluster channels work this way. If they didn’t then a single message at the head of the queue would block all cluster communication if its destination queue manager was unavailable.

Although a single transmission queue for all cluster communication is simple for administrators to understand and is sufficient for many users it does have some drawbacks. Therefore, support for multiple cluster transmission queues was introduced on Windows, Linux and UNIX in version 7.5. IBM i and z/OS did not have a version 7.5 offering so the same feature was introduced in version 8 on these platforms. Many people assume that the introduction of this capability was to improve performance. Some users may notice an improvement if they are constrained by queue contention and/or message buffering but the use of multiple transmission queues predominantly provides the following benefits:

* Separation of message traffic  
  When a single transmission queue is used it is possible for messages destined for one channel to interfere with those for another. For example, if messages cannot be sent over one or more channels then a shared transmission queue can eventually become full.
* Management of messages  
  Administrators often like to use queue attributes such as MAXDEPTH to manage available resources. When all cluster channels share a single transmission queue these attributes become less useful, especially when a queue manager is a member of multiple clusters and the transmission queue is used to service multiple applications.
* Monitoring  
  When a single transmission queue is used it is not possible to use queue monitoring to track the number of messages processed by each channel, although channel statistics provide some of the same information. Administrators must also perform investigative work to identify why the depth of a single transmission queue is growing when it is used by multiple applications and channels. If message traffic is separated it is much easier for administrators to determine the cause and what is affected.

**Configuring multiple cluster transmission queues**

The transmission queue a regular sender channel uses is configured using the XMITQ channel attribute. A similar attribute cannot be used for cluster communication because most channels are automatically defined based on the cluster-receiver definition of each remote endpoint. It would be undesirable and difficult to manage if remote definitions affected the transmission queue local channels use. It might also cause problems for back-level queue managers coexisting in the same cluster. Therefore, an alternative means to configure the transmission queue each cluster-sender channel should use has been implemented.

A new queue manager attribute called DEFCLXQ has been introduced, which stands for ‘default cluster transmission queue’. This attribute has two permissible values, SCTQ and CHANNEL. The value SCTQ, which is the default for backwards compatibility, indicates that by default cluster-sender channels use SYSTEM.CLUSTER.TRANSMIT.QUEUE. The value CHANNEL indicates that by default each cluster-sender channel uses a dynamically created transmission queue called SYSTEM.CLUSTER.TRANSMIT.<channel-name>. Using the value CHANNEL provides administrators with a simple option to use a separate queue for each channel. The queue manager automatically creates and deletes transmission queues as necessary to serve cluster channels.

For many users the use of DEFCLQX alone will be sufficient. However, it is recognised that in large clusters a separate transmission queue for every channel might be too granular. Administrators might also prefer to use a different naming convention for the transmission queues. On z/OS, administrators might wish to control which storage class (page set) and buffer pool is associated with each queue. Therefore, a new queue attribute called CLCHNAME has also been introduced. Instead of defining the transmission queue on the channel, the CLCHNAME attribute allows an administrator to define on a transmission queue which cluster channels should use it. The attribute supports wildcards in any position to allow many channels to use the same manually defined queue. For example, a value of ABC.\* matches any channel with a name that starts with ABC followed by a dot. If the common naming convention of <cluster>.<queue-manager> is used for cluster channels this makes it easy for administrators to configure a separate transmission queue for each cluster a queue manager is a member of, or a separate transmission queue for specific remote queue managers.

The transmission queue a channel uses is determined by searching for a matching CLCHNAME value. If multiple matches are found then the most specific match takes precedence. If no match is found then the value of the DEFCLQX attribute is used to determine which queue to use.

Consider the following example, assuming no other transmission queues have a non-blank CLCHNAME value:

DEFINE QLOCAL(CLUSTER.XMITQ1) USAGE(XMITQ) CLCHNAME(‘AAA.\*’) …  
DEFINE QLOCAL(CLUSTER.XMITQ2) USAGE(XMITQ) CLCHNAME(‘AAA.BBB’) …

* A channel called AAA.BBB uses CLUSTER.XMITQ2 because that transmission queue has a specific CLCHNAME value that matches the name of the channel.
* A channel called AAA.CCC uses CLUSTER.XMITQ1 because that transmission queue has a generic CLCHNAME value that matches the name of the channel and there is not a more specific match.
* The transmission queue used by a channel called XXX.YYY depends on the value of the DEFCLXQ queue manager attribute because no CLCHNAME value matches its name. It will use either SYSTEM.CLUSTER.TRANSMIT.QUEUE or a permanent-dynamic transmission queue called SYSTEM.CLUSTER.TRANSMIT.XXX.YYY.

**Switching transmission queue**

The transmission queue associated with a cluster sender channel can be potentially modified by performing any of the following actions:

* Altering the value of the DEFCLQX queue manager attribute
* Manually defining a transmission queue with a non-blank value for the CLCHNAME attribute
* Altering the value of the CLCHNAME attribute on an existing transmission queue
* Deleting a transmission queue that has a non-blank value for the CLCHNAME attribute

To avoid channels switching transmission queue when they are running, or when multiple configuration changes are made in quick succession, no immediate action is taken by the queue manager when a DEFINE, ALTER or DELETE command is processed. Instead, each channel queries the transmission queue it should use when it starts. If a configuration change has been made since it was last active a switch of its transmission queue is initiated. The process used to switch transmission queue is:

1. The channel opens the new transmission queue for input and starts getting messages from it (using get by correlation ID)
2. A background process is initiated by the queue manager to move any messages queued for the channel from its old transmission queue to its new transmission queue. While messages are being moved any new messages for the channel are queued to the old transmission queue to preserve sequencing. This process might take a while to complete if there are a large number of messages for the channel on its old transmission queue, or new messages are rapidly arriving.
3. When no committed or uncommitted messages remain queued for the channel on its old transmission queue then the switch is completed. New messages are now put directly to the new transmission queue.

Further changes to the transmission queue configuration for a cluster-sender channel do not take effect while the channel is switching, even if the channel is restarted. The existing switch must complete first to avoid messages being dispersed over more than two queues. This is important to remember should you wish to back-out the change that resulted in a switch occurring.

Administrators might not wish cluster-sender channels to switch transmission queue when they next start because this might be at a time when application workload is high. When workload is high there is an inherent race between messages arriving and the queue manager moving them from the old to the new transmission queue in order to complete the switch operation. Although the queue manager will eventually win out CPU consumption (and potentially I/O) will increase during this time. Administrators might also want to avoid a lot of channels switching simultaneously so they can avoid the queue manager spawning many processes to accomplish this. To help avoid this eventuality MQ provides a command that provides the ability to switch the transmission queue of one or more channels that are not running. On distributed platforms the command is called runswchl. On z/OS the CSQUTIL utility can be used to process a SWITCH CHANNEL command instead. Using these commands administrators can explicitly switch one or more channels, either manually or using a script or job. The command processes each channel in turn instead of them all in parallel and waits for each switch to complete before starting the next. This is particularly useful because it avoids administrators having to monitor the status of background switching operations. It is also a good idea to explicitly set the status of the channels that are to be switched to STOPPED beforehand to avoid them being started while the command is running. If a channel is running it will be skipped by the command. Each channel may be started once the switch of its transmission queue has been initiated, even if the moving messages phase has not yet completed. This helps avoid an extended outage for the channel. Messages will be sent by the channel as soon as they have been moved by the queue manager to its new transmission queue.

**Monitoring the status of switch operations**

It is important for administrators to be able to understand the state of systems they manage. To understand the status of switch operations administrators can perform the following actions:

* Monitor the queue manager error log (AMQERR01.LOG) where messages are output to indicate the following stages during the operation:  
  + The switch operation has started
  + The moving of messages has started
  + Periodic updates on how many messages are left to move (if the switch operation does not complete quickly)
  + The moving of messages has completed
  + The switch operation has completed

On z/OS, these messages are output to the queue manager job log, not the channel initiator job log, although a single message is output by a channel to the channel initiator job log if it initiates a switch when starting.

* The DISPLAY CLUSQMGR command can be used to query the transmission queue that each cluster-sender channel is currently using
* The runswchl command (or CSQUTIL on z/OS) can be run in query mode to ascertain the switching status of one or more channels. The output of this command identifies the following for each channel:  
  + Whether the channel has a switch operation pending
  + Which transmission queue the channel is switching from and to
  + How many messages remain on the old transmission queue

This is a really useful command because in one invocation an administrator can determine the status of every channel, the impact a configuration change has had and whether all switch operations have completed.

**Potential issues**

Here is a list of some issues that might be encountered when switching transmission queue, their cause and most likely resolution:

Insufficient access to transmission queues on z/OS

**Symptom:** A cluster-sender channel on z/OS might report it is not authorized to open its transmission queue.

**Cause:** The channel is switching, or has switched, transmission queue and the channel initiator has not been granted authority to access the new queue.

**Resolution:** Grant the channel initiator the same access to the channel’s transmission queue that is documented for the transmission queue SYSTEM.CLUSTER.TRANSMIT.QUEUE. When using DEFCLXQ a generic profile for SYSTEM.CLUSTER.TRANSMIT.\*\* avoids this problem occurring whenever a new queue manager joins the cluster.

Moving of messages fails

**Symptom:** Messages stop being sent by a channel and they remain queued on the channel’s old transmission queue

**Cause:** The queue manager has stopped moving messages from the old transmission queue to the new transmission queue because an unrecoverable error occurred. For example, the new transmission queue might have become full or its backing storage exhausted.

**Resolution:** Review the error messages written to the queue manager’s error log (job log on z/OS) to determine the problem and resolve its root cause. Once resolved, restart the channel to resume the switching process, or stop the channel then use runswchl instead (CSQUTIL on z/OS).

A switch does not complete

**Symptom:** The queue manager repeatedly issues messages that indicate it is moving messages. The switch never completes because there are always messages remaining on the old transmission queue.

**Cause 1:** Messages for the channel are being put to the old transmission queue faster than the queue manager can move them to the new transmission queue. This is likely to be a transient issue during peak workload because if were commonplace then it is unlikely the channel would be able to transmit the messages over the network fast enough.

**Cause 2:** There are uncommitted messages for the channel on the old transmission queue.

**Resolution:** Resolve the units of work for any uncommitted messages, and/or reduce/suspend the application workload, to allow the moving message phase to complete.

Accidental deletion of a transmission queue

**Symptom 1:** Channels unexpectedly switch due to the removal of a matching CLCHNAME value.

**Symptom 2:** A put to a cluster queue fails with MQRC\_UNKNOWN\_XMIT\_Q.

**Symptom 3:** A channel abnormally ends because its transmission queue does not exist.

**Symptom 4:** The queue manager is unable to move messages to complete a switch operation because it cannot open either the old or the new transmission queue.

**Cause:** The transmission queue currently used by a channel, or its previous transmission queue if a switch has not completed, has been deleted.

**Resolution:** Redefine the transmission queue. If it is the old transmission queue that has been deleted then an administrator may alternatively complete the switch operation using runswchl with the -n parameter (or CSQUTIL with MOVEMSGS(NO) on z/OS). Use the -n parameter with caution because if it is used inappropriately then messages for the channel can be orphaned on the old transmission queue. In this scenario it is safe because as the queue does not exist there cannot be any messages to orphan.

**Summary**

I hope you’ve found this blog post useful. It has described how to use multiple cluster transmission queues, some benefits of doing so and how to switch the transmission queue used by cluster channels. Some potential issues that might be encountered have also been discussed along with their most likely cause and resolution. For more information about this capability see IBM Knowledge Center or the “*WebSphere MQ V7.1 and V7.5 Features and Enhancements*” RedBook, which is available [here](http://www.redbooks.ibm.com/redbooks.nsf/RedpieceAbstracts/sg248087.html). A good starting point in IBM Knowledge Center is the "*Clustering: Best Practices*" topic, which is available [here](http://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.pla.doc/q004740_.htm).