Maximising WebSphere MQ availability in low-latency environments

Jamie Halstead

March 25, 2009

Certain parameters of WebSphere MQ channel objects can prove unexpectedly problematic, as their default settings can exaggerate the duration of availability outages. This article shows you how to avoid this problem through careful tuning of the channel objects, which is especially important when your environment requires low latency and high availability. This article is based on field observations of unexpected, undesirable, and obscure behaviours with WebSphere MQ.

Introduction

Whilst the default IBM® WebSphere® MQ channel object configurations are usually acceptable, a number of settings have proven problematic in the field. A fundamental benefit of messaging is that it decouples systems so that they can work independently and asynchronously; if a node experiences an outage, the messaging software will store messages and forward them on when service is restored. The WebSphere MQ defaults that determine the speed with which any stored messages are forwarded after an outage are reasonable for many situations, but they need to be revised for low latency solutions. This article describes three areas where significant time can be lost when using the default configuration:

- Basic channel retry settings -- 60 to 1200 second availability losses in the default configuration
- Detecting that the target node has become unavailable -- Up to 360 second availability loss in the default configuration
- Message retry pauses associated with transient undeliverable messages -- Up to 500 second availability loss in the default configuration

Whilst this article is focussed on availability for low-latency environments, these settings should always be considered when implementing WebSphere MQ. In many SOA implementations, challenging non-functional requirements are specified around quality of service, throughputs, and near real-time processing. To meet these demands, WebSphere MQ is often used as the messaging service provider for the ESB when assured delivery is required, so this article is also useful whenever WebSphere MQ forms a foundational component of your SOA solution.
Basic channel retry settings

60 to 1200 second availability losses in the default configuration

Channels are the foundation of WebSphere MQ intercommunication, enabling messages to be passed from one queue manager to another with an assurance of once-only delivery from source to target. The advent of WebSphere MQ clustering has hidden many of the complexities of distributed queue management, but the fundamental underlying objects are still the same as in traditional WebSphere MQ networks.

Often when channels are defined, only the most significant attributes are specified, such as the name, channel type, connection name, and transport type. Significant faith is placed in the defaults that will be inherited from the SYSTEM.DEF.<channel type> objects.

Channel retry functionality enables WebSphere MQ to recover from outages in a partner queue manager, whether these are brought about by network problems, administrative or operational procedures, or hardware failures. Channel retry attempts are governed by the four parameters described below:

Table 1. Channel parameters affecting connection retry

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Retry Interval</td>
<td>SHORTTMR 60 sec</td>
<td>Defines the number of seconds that the channel will delay before next attempting to re-establish communications with its partner.</td>
</tr>
<tr>
<td>Short Retry Count</td>
<td>SHORTRTY 10</td>
<td>Defines the maximum number of short retry attempts a channel will perform to re-establish the session with its partner; each short retry will take place after the interval specified in the short retry interval attribute.</td>
</tr>
<tr>
<td>Long Retry Interval</td>
<td>LONGTMR 1200 sec</td>
<td>Defines the number of seconds that the channel will delay before next attempting to re-establish communications with its partner.</td>
</tr>
<tr>
<td>Long Retry Count</td>
<td>LONGRTY 999,999,999</td>
<td>Defines the maximum number of attempts a channel will execute when trying to re-establish the session with its partner. Long retry processing starts once the short retry count has been exhausted. Each long retry will take place after the interval specified in the long retry interval attribute.</td>
</tr>
</tbody>
</table>

If the initial connection allocation attempt fails or a problem is encountered during operations, a reconnection is attempted every Short Retry Interval up to the Short Retry Count number of times. If communication has not been re-established after the defined number of Short Retry Count attempts, a long retry cycle will be initiated, attempting connection every Long Retry Interval up to the Long Retry Count number of times.
Therefore, in the default configuration, if the receiver channel is no longer available (for example, if the host machine crashed) the sender channel will try to reconnect every minute for the first ten minutes, after which it will attempt reconnection every 20 minutes. In low-latency environments where time critical information is being transmitted or aggressive service level agreements need to be met, allowing a communication path to remain down for up to 20 minutes beyond the end of the outage is unacceptable. In this instance the default parameters magnify the duration of the outage and extend the recovery time once transmission restarts.

To put this point in perspective, consider Example 1: Assume that the affected channel transfers an average of 50 messages per second and a network problem inhibits communications for 11 minutes and 15 seconds. With the standard channel configuration based on the defaults, it will be 30 minutes before the channel restarts and there will be 90,000 messages awaiting transmission and subsequent processing. The depth of the transmission queue and the points at which channel connection retry attempts are made are illustrated in Graph 1:

**Graph 1. Default Channel Configuration, Retry attempts and transmission backlog**

![Graph 1](image)

Table 2 shows the settings for a slightly more appropriate configuration:

**Table 2. Channel Retry settings to enable rapid resumption of communications**

<table>
<thead>
<tr>
<th>Channel Attribute</th>
<th>Configured Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Retry Interval <strong>SHORTTMR</strong></td>
<td>10 seconds</td>
</tr>
<tr>
<td>Short Retry Count <strong>SHORTRTY</strong></td>
<td>60</td>
</tr>
<tr>
<td>Long Retry Interval <strong>LONGTMR</strong></td>
<td>30 seconds</td>
</tr>
<tr>
<td>Long Retry Count <strong>LONGRTY</strong></td>
<td>999,999,999</td>
</tr>
</tbody>
</table>

By applying these settings, communications will now restart after 11 minutes and 30 seconds and there will be only 34,500 messages awaiting transmission and subsequent processing. The depth of the transmission queue and the points at which channel reconnection attempts are made are illustrated in Graph 2:
Graph 2. Revised Channel Configuration

As shown, 18 minutes and 30 seconds of unwarranted and undesired inactivity have been avoided, and the size of the resulting backlog has been reduced 55,500 messages or 62%.

As always, you need to strike a balance between the availability requirements of the solution and the cost, which in this case is the cost of spending excessive processor time trying to re-establish communications to a node experiencing a prolonged outage. Settings such as those suggested in Table 2 above have been successfully implemented in production systems without any increase in processor time. From an implementation perspective, these parameter changes will be adopted only when the channels they are applied to negotiate their connection on start up, which can be achieved by stopping and starting the associated sender channel instance.

Detecting that the target node has become unavailable

Up to a 360 second availability loss in the default configuration

Nodes inevitably go down from time to time, so the impact on WebSphere MQ and the speed of detection are crucial. Once the queue manager detects that its channel connection is broken, it can initiate channel retry processing. If WebSphere MQ clustering is being used, pending work will be rerouted only to alternate queue instances throughout the cluster once the channel is in a non-running state. If the target queue manager goes down, the behaviour of the sending queue manager is controlled by the heartbeat interval parameter HBINT:

Table 3. Heartbeat interval parameter

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart Beat Interval</td>
<td>300</td>
<td>Specifies the approximate time between heartbeat flows that are to be passed from a sending MCA (Message Channel Agent) when there are no messages on the transmission queue. The value must be in the range 0 to 999,999. A value of zero means that no heartbeat flows are to be sent.</td>
</tr>
</tbody>
</table>
When the node hosting the target queue manager goes down, the sender queue manager's sender channel continues attempting to send messages to the target queue manager and remains in a running state until a timeout occurs. Therefore, from a WebSphere MQ perspective, the problem is not known and no retry processing is initiated. From a clustering perspective, the cluster workload balancing algorithm will still route traffic to any cluster queues that exist on the failed queue manager. The period of time a channel waits to timeout is determined by the HBINT parameter and is calculated as shown in Table 4 below. The default 6-minute timeout has proven both problematic and undesirable in environments where low latency and high availability are required.

**Table 4. Channel timeout calculation**

<table>
<thead>
<tr>
<th>HBINT Value</th>
<th>Timeout calculation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greater than or equal to 60</td>
<td>HBINT + 60</td>
<td>HBINT = 300. Timeout = 360s.</td>
</tr>
<tr>
<td>Less than 60</td>
<td>HBINT x 2</td>
<td>HBINT = 15. Timeout = 30s.</td>
</tr>
</tbody>
</table>

When WebSphere MQ recognises that the channel has failed, the queue manager automatically performs a reallocation of those messages on the transmission queue that were about to go through this channel. If possible, it reroutes the message to another instance of the target queue, though only in cases where cluster queues are being accessed and opened in a non-fixed bind mode. The sooner this reallocation can be performed and the non-running state detected, the better, in order to prevent the message being delayed.

Changing the HBINT parameter has a number of consequences. If no messages are flowing through a channel, a heartbeat flow will be initiated with the frequency of the HBINT parameter. This flow generates a slight network overhead (28 bytes), and in addition the channel pooling processes (amqrmppa) and the queue manager agent processes (amqzlaa0) will consume additional CPU resources. However, this overhead has not proved significant or even noticeable when the heartbeat has been reduced to as little as 10 seconds.

The HBINT attribute of both the sender and receiver channels needs to be changed and ideally aligned, because at channel startup, a negotiation is performed between both ends and the setting that demands the lower resources is selected (the longer heartbeat). From an implementation perspective, the settings must be changed at both sender and receiver end, after which the sender channel can be stopped and started to force a renegotiation.

**Message retry pauses associated with transient undeliverable messages**

**8 minute availability loss in the default configuration**

When WebSphere MQ sends messages between queue managers, it uses channels. Regardless of type, these use components called Message Channel Agents (MCA), which are responsible for interaction between the queue manager and the communication links between them. Here is a typical distributed queue configuration with sender and receiver channels:
In this scenario, the key functionality is the responder MCA -- the MCA associated with the receiving queue manager. It receives a batch of messages from the caller MCA and, based on transmission information, resolves which local queue to deliver the messages to. All messages in the batch are put to their respective queues before a subsequent batch is processed.

There are a variety of cases in which the put to the destination queue might fail. The behaviour under consideration here is transitory failures -- queue full or put disabled. In the default channel configuration, when a transitory failure is encountered, the MCA will wait 1000ms and retry the put operation 10 times -- in other words the message retry operation could last for 10 seconds. If the transitory failure has not resolved itself over this period, the message will be delivered to the Dead Letter Queue on the destination queue manager. As all messages in a batch (up to 50 by default) are processed serially, this procedure can lead to significant delays. If all 50 messages were directed to the same queue it would lead to:

- A 500-second delay before the batch completes and messages are available for processing on the queue manager
- A significant backlog of messages at the sender end
- Increased probability of message loss if expiry is being used

The behaviour seen at the receiving application in terms of message arrival will be different for persistent and non-persistent messages. If the NPMSPEED parameter is left at its default (FAST) on the channels, then non-persistent messages are delivered to the target queue straightaway and do not wait until the batch completes to be committed. There are three possible permutations:

- **Persistent Messages** -- All persistent messages are committed to the target queues in a single unit of work upon completion of the batch.
- **Non Persistent Messages and NPMSPEED set to NORMAL** -- All messages regardless of persistence are committed to the target queues in a single unit of work upon batch completion.
- **Non Persistent Messages and NPMSPEED set to FAST** -- All non-persistent messages are committed to their queue as soon as the put completes regardless of batch completion.

To illustrate this situation, consider Example 2, in which an extremely low queue depth is configured to force a queue full situation, which triggers the message retry processing under...
examination. There are two queue managers JEREMY and KEN. JEREMY is the sender whilst KEN is the receiver hosting queues IN001, IN002, and IN003. IN001 and IN002 have a maximum depth of 5000, while IN003 has a maximum depth of 1. The JEREMY queue manager has three remote queues (IN1, IN2, and IN3) configured to point to the correspondingly numbered queues on the KEN queue manager. Sender and Receiver channels named JEREMY.TO.KEN are configured on both queue managers, and the default message channel retry and message channel interval settings are accepted. Here is the configuration:

**Figure 2. Diagram of test setup**

To explore the behaviour of the channel message retry processing when a transient failure occurs on the receiving end, the following steps are performed:

- The sender channel JEREMY.TO.KEN is stopped on queue manager JEREMY.
- Fifty messages are put on the sender queue manager in a round robin fashion, in order IN001, IN002, IN003, IN001, IN002 ...
- The sender channel JEREMY.TO.KEN is started on queue manager JEREMY.

In the case where persistent messages are sent, or the NPMSPEED is NORMAL, they are available only to consuming applications on the KEN queue manager once the entire batch of messages has been transmitted. The first five messages are delivered almost instantly, but message #6 for queue IN003 cannot be delivered because IN003 has a maximum depth of 1 and is full. Therefore message retry processing is invoked. Ten attempts to deliver the message to IN003 are made, each 1 second apart, until finally this message is delivered to the dead letter queue DLQ. Messages #7 and #8 can be delivered, but message #9 for queue IN003 cannot be delivered as it is full and message retry processing is again invoked. This pattern continues until the batch of 50 messages is processed and in total this batch takes 2.5 minutes to be transmitted. The arrival of persistent messages on the target queue manager is shown in Graph 3:
As discussed the messages are only available for processing after the entire batch completes, as shown in the graph. During the 150 seconds that it took for this one batch of messages to be delivered, a large volume of messages will have no doubt built up on the sender queue manager.

When non-persistent messages are sent and NPMSPEED is FAST, messages that are delivered to the queues on the KEN queue manager are accessible to any consuming application immediately. The first five messages are delivered almost instantly, but message #6 for queue IN003 cannot be delivered as it is full. Message retry processing is invoked and after 10 seconds the message is routed to the dead letter queue DLQ. Message #7 and #8 can be delivered, but message #9 for queue IN003 cannot be delivered as it is still full. Effectively a 10 second pause in message delivery takes place every third message. The arrival of non-persistent messages on the target queue manager is illustrated in Graph 4:

In this case a delay of 150 seconds is still seen, but because delivery to target queue is not tied to a batch-wide unit of work, messages that can be delivered make it to queue as soon as possible.
and can be processed by applications. The 10-second delay introduced every time a message is bound for the IN003 queue can clearly be seen.

By making a few simple changes to the WebSphere MQ receiver channel configuration the impact of this retry functionality can either be minimised or completely disabled. The parameters that control this behaviour are defined on the receiver channels as shown in Table 5:

**Table 5. Channel parameters affecting message retry**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Retry Interval MRTMR</td>
<td>1000</td>
<td>The minimum time in milliseconds that must pass before the channel can retry the MQPUT operation. A value of 0 means that the retry will happen as soon as possible.</td>
</tr>
<tr>
<td>Message Retry Count MRRTY</td>
<td>10</td>
<td>The number of times the channel will retry before it decides it cannot deliver the message. A value of 0 means that the retry functionality is disabled.</td>
</tr>
</tbody>
</table>

In many implementations, the life of a transaction or message can be short, on the order of seconds. Therefore any message retry may need to be reduced to the bare minimum necessary to meet the non-functional requirements, and recommended values cannot be given. In different engagements, based on the particular circumstances, I have implemented channel configurations which:

- Deactivated Message Channel Retry Completely
- Used MRTMR and MRRTY values in the low single figures
- Stuck with the product defaults

A reasonable set of MRTMR and MRRTY parameters can be calculated, based on the shortest anticipated life of a message and highest throughput of messages across a channel. The aim is to give a balanced result in terms of allowing retries to take place without performing excessive MQI interaction or significantly delaying other WebSphere MQ traffic. From an implementation perspective, these changes will be required on all receiver channel instances, and for the configuration changes to take effect, the channels must be recycled at a suitable point in time. This can be achieved by stopping and starting the associated sender channel instance, which puts the receiver into inactive status.

**Additional WebSphere MQ parameters**

You can tune additional WebSphere MQ parameters to affect the transfer of messages between queue managers. These parameters are indirectly related to the article sections Detecting that the target node has become unavailable and Message retry pauses associated with transient undeliverable messages, and can be used to further refine system behaviour to meet your requirements, as shown in Table 6:

**Table 6. Additional WebSphere MQ parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
### Conclusion

This article has described three potential problems with the default WebSphere MQ Channel configuration. In summary, you should consider the following questions when planning or reviewing your channel infrastructure:

- What is the maximum acceptable amount of time that communication between queue managers can be down when all resources necessary are available?
- What is the maximum acceptable amount of time that channels can take to time out when attempting to communicate with a partner queue manager?
- What is the maximum acceptable delay to a message being passed between queue managers?

Use the answers to these questions in combination with the information in this article to configure WebSphere MQ channels to provide the required levels of availability and reduce latency.

**A final note of caution:** Changing the parameters described in this article can improve WebSphere MQ performance in your environment, but as with all changes, you should test them thoroughly to ensure that they deliver benefits and do not cause other problems.
Related topics

- **WebSphere MQ Fundamentals**
  This IBM Redbook is an excellent first stopping point to build or refresh your technical knowledge of WebSphere MQ. Topics include message queuing, MQ applications, queue managers, intercommunication, queue manager clusters, messaging, security, and more.

- **WebSphere MQ Intercommunication for V7**
  In WebSphere MQ, intercommunication means sending messages from one queue manager to another. The receiving queue manager could be on the same machine, or on another machine nearby or across the world. This book takes you through intercommunication concepts and WebSphere MQ intercommunication facilities.

- **WebSphere MQ developer resources page**
  Technical resources to help you design, develop, and deploy messaging middleware with WebSphere MQ to integrate applications, Web services, and transactions on almost any platform.

- **WebSphere MQ product page**
  Product descriptions, product news, training information, support information, and more.

- **WebSphere MQ V7 trial download**
  A no-charge trial download of WebSphere MQ V6. Includes limited online support for Windows® and Linux® installations at no charge during the trial period.

- **WebSphere MQ V6 information center**
  A single Web portal to all WebSphere MQ V6 documentation, with conceptual, task, and reference information on installing, configuring, and using your WebSphere MQ environment.

- **WebSphere MQ documentation library**
  WebSphere MQ product manuals.

- **WebSphere MQ support page**
  A searchable database of support problems and their solutions, plus downloads, fixes, problem tracking, and more.

- **WebSphere MQ public newsgroup**
  A non-IBM forum where you can get answers to your WebSphere MQ technical questions and share your WebSphere MQ knowledge with other users.

- **WebSphere MQ SupportPacs**
  Downloadable code, documentation, and performance reports for the WebSphere MQ family of products.

- **developerWorks WebSphere Business Integration zone**
  For developers, access to WebSphere Business Integration how-to articles, downloads, tutorials, education, product info, and more.

- **WebSphere Business Integration products page**
  For both business and technical users, a handy overview of all WebSphere Business Integration products.

- **Most popular WebSphere trial downloads**
  No-charge trial downloads for key WebSphere products.

- **Technical books from IBM Press**
  Convenient online ordering through Barnes & Noble.