Enhanced WebSphere MQ cluster workload balancing with Omegamon XE for Messaging and IBM Tivoli Monitoring

Ian Vanstone

Introduction

IBM® WebSphere® clustering provides a powerful set of message workload balancing features. By default, messages are load balanced based on queue manager availability, but they are not load balanced based on the availability of message consumer applications. Therefore messages can be sent to cluster queues that are not being actively served by message consumer applications. WebSphere MQ triggering can be used to avoid messages remaining unprocessed on queues, but sometimes triggering is not suitable, such as when applications have special connection pooling or startup logic. If triggering is not enabled and messages are left unprocessed on queues, there is an increased risk of service failures or timeouts.

This article shows you how to send messages only to those queues that are being actively served by message consumer applications. The solution reduces failures and service timeouts caused by outages of WebSphere MQ applications that are not suitable for triggering. The solution uses:

- WebSphere MQ clustering to workload balance messages
- Omegamon XE for Messaging and IBM Tivoli Monitoring to monitor the state of WebSphere MQ queues and to automate configuration changes

Why workload balancing based on application availability?

WebSphere MQ clustering does not load balance messages based on the availability of message consumer applications, as shown in Figure 1 below with a cluster of three queue managers. The message producer application connected to QM1 puts messages to queue CLUSQ1. By default, clustering load balances messages equally between the instances of CLUSQ1 on QM2 and QM3, even though CLUSQ1 on QM3 is not being served by a message consumer application:
In most environments, if messages arrived on CLUSQ1 on QM3, WebSphere MQ triggering could be used to start a message consumer application. In this example, the message consumer application runs within an application server that is offline due to an unplanned outage. Because of special startup procedures, triggering cannot be used to start the application server. In this situation, whilst the application server is offline, it is better to send all messages to CLUSQ1 on QM2, which has an active application server and message consumer application. The requirement to send messages to queues with active message consumer applications is relevant in a number of situations, including where:

- A service level agreement (SLA) specifies maximum message processing times
- A service timeout has been configured in one or more of the service components
- A person initiated the service and must wait for a reply, and will eventually give up waiting

You can architect workload balancing based on application availability using features of WebSphere MQ along with additional products -- in this case, IBM Tivoli Monitoring and Omegamon XE for Messaging.

**Products used**

This section describes how the three products are used to load balance messages based on application availability. For a discussion on other solutions, see Alternatives below.

**WebSphere MQ Clustering**

WebSphere MQ clustering is used to combine queue managers into groups. As demonstrated in the solution, clustering serves two main purposes:

1. Clustering provides a message workload balancing system, typically used to spread messaging workloads across multiple queue managers or to send messages to the queue
managers with highest availability. The solution in this article enhances the workload balancing features.

2. Clustering provides an object auto-definition system, enabling the creation of WebSphere MQ messaging networks that are flexible during changes in topology. The result is a simpler administration model. For example, adding or removing a queue manager from a cluster is simpler than doing so from a large WebSphere MQ network interconnected with regular (non-cluster) channels. Clustering is increasingly used in the SOA messaging paradigm, which requires that multiple queue managers connect directly to one another to provide a flexible enterprise service bus (ESB). Cluster auto-definition features are used in the solution to automatically share object definition changes between cluster queue managers.

**Omegamon XE for Messaging**

Omegamon XE for Messaging provides monitoring and configuring features for WebSphere MQ and WebSphere Message Broker. In this solution, it is used to monitor the status of WebSphere MQ queues.

**IBM Tivoli Monitoring**

Tivoli Monitoring provides infrastructure and tools to monitor, configure, and automate the running of healthy business systems. Many other monitoring products can be connected to Tivoli Monitoring to provide a centralized enterprise-wide monitoring solution. This solution uses Omegamon XE for Messaging in conjunction with Tivoli Monitoring to monitor, configure, and automate changes to WebSphere MQ assets. The Tivoli Enterprise Portal graphical user interface is used to configure features of Omegamon XE for Messaging and to automate operations. Tivoli Monitoring provides alternative administration interfaces to the Tivoli Enterprise Portal.

**Solution architecture**

The solution includes three key stages to workload balancing based on application availability:

**Detecting the health of applications**

The open input count (IPPROCS) queue attribute is used to indicate the presence of message consumer applications. IPPROCS shows the number of applications that have the queue open for input. A non-zero open input count indicates that at least one application is ready to process messages on the queue. A zero open input indicates that there are no applications ready to process messages on the queue.

**Influencing the cluster workload balancing algorithm**

The put (PUT) queue attribute is used to influence the cluster workload balancing. The usefulness of PUT in the solution is based on two powerful features of clustering. Firstly, the cluster workload algorithm does not send messages to put-disabled queues. Secondly, clustering automatically propagates cluster queue object definitions when their put attribute is altered. Using Figure 1 above as an example, if CLUSQ1 on QM2 is altered to be put-disabled, the object change is automatically propagated to QM1. Therefore messages put by the application connected to QM1 will not be sent to QM2, and hence, all messages will be sent to QM3. If CLUSQ1 on QM2 is then subsequently altered to be put-enabled, again the object change is automatically propagated to QM1, resulting in the message workload being balanced equally between QM2 and QM3.
Ensuring that messages are processed in a timely manner

If messages are queued on a cluster queue that lacks active message consumer applications, it is useful to redistribute the messages so that they are sent to queues with active message consumer applications. The solution uses a simple application to get each message from the cluster queue with no active message consumer applications and put them to another cluster queue. The cluster workload balancing algorithm does not allow puts to queues that are put-disabled (those queues with no active message consumer applications). In this way, messages are redistributed to queues where they are most likely to be processed in a timely manner.

Solution components

Key solution components are listed below and shown in Figure 2:

- Three WebSphere MQ cluster queue managers:
  - QM1 has a message producer application connected, putting messages to cluster queue DCQ1.
  - QM2 and QM3 each host an instance of the cluster queue DCQ1. The solution ensures that messages are sent to only those instances that have active message consumer applications.
  - QM2 and QM3 are monitored with Omegamon XE for Messaging agents.
- Two shell scripts:
  - The put-enable script, used to put-enable the queue.
  - The put-disable script, used to put-disable the queue. This script also executes the Q application (available in WebSphere MQ SupportPac MA01) to redistribute messages. The Q application is simple and easy to use and also useful for testing.
- Tivoli Monitoring components are connected to the Omegamon XE for Messaging agents and viewed using Tivoli Enterprise Portal from a Web browser running on a laptop. Tivoli Enterprise Portal is used to configure situations that monitor the open input count of queues and automatically execute the shell scripts, such that:
  - If the open input count is zero, the put-disable script is executed.
  - If the open input count is non-zero, the put-enable script is executed.
**Configuring the solution**

The general principles described in this article can be adapted for many product and platform combinations. This particular solution was deployed and tested using the following AIX V5.3 software:

- Tivoli Monitoring V6.2.1
- Omegamon XE for Messaging V6.0.1.1
- WebSphere MQ V6.0.2.5

These configuration instructions make the following assumptions:

- Omegamon XE for Messaging and Tivoli Monitoring are pre-configured, so that queue managers can be monitored in the Tivoli Enterprise Portal.
- Three queue managers (QM1, QM2, and QM3) are connected in a WebSphere MQ cluster.
- Cluster queue DCQ1 is defined on both QM2 and QM3 with the DEFBIND queue attribute set to NOTFIXED. For example: `DEF QL(DCQ1) CLUSTER(C1) DEFBIND(NOTFIXED)`

**Set up the shell scripts**

1. Store the following shell scripts onto Machine2 and Machine3 in directory `/opt/MQScripts`. 

---

Enhanced WebSphere MQ cluster workload balancing with Omegamon XE for Messaging and IBM Tivoli Monitoring
• queueEnable.sh put-enables the queue.

```sh
#!/bin/sh
#
# Check that the required parameters were used
if [ $# -ne 2 ]
then
    echo "$0 : You must supply the queue and qmgr"
    exit 1
fi

# Put-enable the queue
echo 'ALTER QL(' $1 ') PUT(ENABLED)' | runmqsc $2
```

• queueDisable.sh put-disables the queue and redistributes the messages on the queue.

```sh
#!/bin/sh
#
# Check that the required parameters were used
if [ $# -ne 2 ]
then
    echo "$0 : You must supply the queue and qmgr"
    exit 1
fi

# Put-disable the queue
echo 'ALTER QL(' $1 ') PUT(DISABLED)' | runmqsc $2

# Redistribute any messages on the queue by
# using the Q application
/opt/IBM/MA01/q -I$1 -m$2 -o$1 –p1
```

This solution uses shell scripts so that the WebSphere MQ configuration and execution of the message redistribution application are carried out together in a single place, and in a consistent manner. Alternatively, you can do this using Tivoli Enterprise Portal workflows and Omegamon XE for Messaging.

2. Download WebSphere MQ SupportPac MA01 to both Machine2 and Machine3, storing the Q application in the /opt/IBM/MA01 directory.

**Set up the Tivoli Enterprise Portal situations**

1. Start a Web browser and log on to Tivoli Enterprise Portal.
2. Click on the **Situations Editor icon** ( ) on the toolbar:

   **Figure 3**

   ![Tivoli Enterprise Portal](image)

3. Expand the **MQSERIES** section in the Situation Editor window:
4. Right click on MQSERIES and select **Create New** to begin creating the situation associated with cluster queues that do not have applications ready to process messages.

5. Enter the name, description, and type of the new situation:

**Figure 5**

6. In the Select Condition window, select the **Queue Data** attribute group, and the **Input Opens** and **Queue Name** attribute items. Then click **OK**.
7. Select the **Formula** tab in the Situation Editor window. Set the formula so that the situation fires if no applications have the DCQ1 queue open for input:
   
   1. Set the Queue Name value to: `== 'DCQ1'`
   2. Set the Input Opens value to: `== 0`
   3. Set the Sampling interval to: 5 minutes

   Set the sampling interval based on maximum message processing times.
8. Select the Distribution tab in the Situation Editor window. Add the appropriate queue managers to the assigned list. In this example, the `DCQ1` queue is hosted on the `QM2` and `QM3` queue managers.

9. Select the Action tab in the Situations Editor window:
   1. Set the System Command value to:
      
      ```bash
      /opt/MQScripts/queueDisable.sh
      &{Queue_Data.Queue_Name}
      &{Queue_Data.MQ_Manager_Name}
      ```
   2. Set the Take action on each item radio button, so that the situation fires for each queue that meets the formula requirements.
   3. Set the Execute the Action at the Managed System (Agent) radio button, so that the system command executes on the correct machine (the machines hosting the scripts).
   4. Set the Don't take action twice in a row (wait until situation goes false then true again) radio button, so that the scripts are only run when the state of the application changes.
Figure 8

10. Click **Apply** to save the situation.
11. Expand the **MQSERIES** section in the Situation Editor window.
12. Right click on MQSERIES and select Create New to begin creating the situation associated with cluster queues which have applications processing messages.

13. Enter the name, description, and type of the new situation:

**Figure 10**

![Create Situation Window]

14. In the Select condition window, select the Queue Data attribute group and the Input Opens and Queue Name attribute items. Then click OK.
15. Select the **Formula** tab in the Situation Editor window. Set the formula so that the situation fires if any applications have the DCQ1 queue open for input:
   1. Set the Queue Name value to: `== 'DCQ1'`
   2. Set the InputOpens value to: `> 0`
   3. Set the Sampling interval to: 5 minutes

Set the sampling interval based on maximum message processing times:
16. Select the **Distribution** tab in the Situations Editor window. Add the appropriate queue managers to the assigned list. In this example, the `DCQ1` queue is hosted on the `QM2` and `QM3` queue managers.

17. Select the **Action** tab in the Situations Editor window.
   1. Set the System Command value to:
      
      ```
      /opt/MQScripts/queueEnable.sh
      &{Queue_Data.Queue_Name}
      &{Queue_Data.MQ_Manager_Name}
      ```
   
   2. Set the **Take action on each item** radio button, so that the situation fires for each queue that meets the formula requirements.
   3. Set the **Execute the action at the Managed System (Agent)** radio button, so that the system command executes on the correct machine (the machines hosting the scripts, local to the queue managers).
   4. Set the **Don't take action twice in a row** (wait until situation goes false then true again) radio button, so that the scripts are only run when the state of the application changes.
18. Click **Apply** to save the situation.
19. Right click on the **MQQueueEnable** situation and click **Start Situation**.
20. Right click on the **MQQueueDisable** situation and click **Start Situation**.
21. Click **OK** to exit the Situation Editor.

The configuration is now complete and ready for testing.

**Testing the solution**

**Summary of test:**

1. Connect message consumer applications to each cluster queue DCQ1 and check that both queues are put-enabled.
2. Put messages to the DCQ1 queues using a message producer application connected to QM1.
3. Stop one of the message consumer applications and put more messages to the DCQ1 queues, using the message producer application. Messages are then queued on the instance of DCQ1 without an active message consumer application.
4. Check that once the situation fires, the instance of DCQ1 without a message consumer application is put-disabled, and its messages are redistributed to the instance of DCQ1 with an active message consumer application.

The test application is the Q application from WebSphere MQ SupportPac MA01. To test the solution, follow these steps:

1. **Download WebSphere MQ SupportPac MA01** to Machine1, storing the Q application in the /opt/IBM/MA01 directory.
2. Start the message consumer applications so they can get messages from the instances of DCQ1:
   1. Run the following command on Machine2: `/opt/IBM/MA01/ q -IDCQ1 –mQM2 -w10000`
   2. Run the following command on Machine3: `/opt/IBM/MA01/ q -IDCQ1 –mQM3 -w10000`
3. Start a message producer application on QM1 to put messages to DCQ1:
   1. Run the following command on Machine1: `/opt/IBM/MA01/ q -oDCQ1 -mQM1`
4. Wait the duration specified in the situation sampling interval in Tivoli Enterprise Portal, and then check the put attributes of the cluster queues from QM1. Run the following command on Machine1 and check that both instances of DCQ1 have their PUT attribute set to ENABLED:
   ```
   echo 'DIS QC(DCQ1) ALL' | runmqsc QM1.
   ```
5. Put 100 messages to queue DCQ1 using the message producer application connected to QM1. Run the following command in the Q application session on Machine1: `#100`
6. Check that the messages arrived on QM2 and QM3. Ensure that text (for example, (4 bytes) #100) appears in the Q application sessions on both Machine2 and Machine3.
7. Kill the message consumer application on Machine2 by typing Ctrl + C:
   1. Put another 100 messages to queue DCQ1 using the message producer application connected to QM1. Run the following command in the Q application session on Machine1: `#100`
   2. Immediately check that messages are now queued on DCQ1 on QM2. Run the following command on Machine2 and check that the DCQ1 queue has CURDEPTH greater than zero: `echo 'DIS QL(DCQ1) CURDEPTH' | runmqsc QM2`
8. Wait the duration specified in the situation sampling interval in Tivoli Enterprise Portal (for example, 5 minutes), then check the put attributes of the cluster queues from QM1. Run the following command on Machine1 and check that the QM2 queue has PUT set to DISABLED and that the QM3 queue has PUT set to ENABLED: `echo 'DIS QC(DCQ1) ALL' | runmqsc QM1`
9. Also check that the messages on QM2 have been redistributed. Run the following command on Machine2 and check that the DCQ1 queue has CURDEPTH equal to zero: `echo 'DIS QL(DCQ1) CURDEPTH' | runmqsc QM2`

The testing is now complete.

**Alternatives**

The solution includes three stages to workload balancing based on application availability:

1. Detecting the health of applications
2. Influencing the cluster workload balancing algorithm
3. Ensuring that messages are processed in a timely manner

This section explains how to implement each stage using alternative approaches.

**Detecting the health of applications**

You can use the open input count to approximate which queues have applications ready to process messages, but it is not always reliable. If a queue has a non-zero open input count, an application may not be ready to process messages. Possible causes include:

- The application has the queue open, but is otherwise unhealthy or not prepared to process messages (for example, it has run out of worker threads).
- The application does not have the capacity to process the message rates defined in the relevant service level agreement.
- The machine itself is unhealthy (for example, the CPU is already running at 100% utilization).
- The wrong application is connected to the queue.

Alternatives to using the open input count:

- Use the current depth of the queue. If the queue depth is high, assume that the application is not healthy or able to cope with the rate of messages. Queue depth events could be used to initiate configuration changes.
- Monitor the actual application process. If the process is running, assume that the application is healthy.
- Add an application interface so that it can be polled. If the call to the interface returns a good return code, assume that the application is healthy.
- These alternatives can be combined, and they can be fully or partly implemented using features of Omegamon XE for Messaging and Tivoli Monitoring.

**Influencing the cluster workload balancing algorithm**

Put-enabling and put-disabling queues is a good approach because clustering will automatically propagate changes to the put attribute and use the attribute when workload balancing. Cluster workload rank (CLWLPRANK) is another queue attribute that fits this automatic change propagation model. The cluster workload rank attribute differs from the put attribute in two ways:

- It is more granular. Rather than holding values of enabled and disabled, the cluster workload rank value ranges from 0 to 9. Messages are sent to the highest ranked queue or queues, and so this granularity allows queues to be ordered to indicate grades of application availability.
- It does not cause puts to fail in situations where all queues have equal low values. If a message is put to a queue where all queue instances are put-disabled, the put fails and the putting application receives a bad return code (MQRC_CLUSTER_PUT_INHIBITED). If a message is put to a queue where all queue instances are equally ranked, the put to the queue completes successfully and the message workload is balanced evenly across all queue instances.
If you use cluster workload rank, set cluster workload use queue (CLWLUSEQ) to ANY so that the message workload is balanced to remote queues even if a local queue exists.

The difference in behaviour between the put attribute and the cluster workload rank attribute provide two classes of service:

- Use the put attribute if you want puts (to queues where no instance of the queue has an active application) to fail.
- Use the cluster workload rank attribute if you want puts (to queues where no instance of the queue has an active application) to complete and messages to be queued.

There are additional alternatives, including altering cluster channel attributes that influence the cluster workload algorithm (for example, cluster workload rank or cluster workload weight).

### Ensuring that messages are processed in a timely manner

In the solution described in this article, the Q application is used to redistribute messages to queues that have active applications. You must ensure that messages are redistributed correctly. In situations where all instances of a cluster queue are put-disabled, the redistribution application will fail (because of MQRC_CLUSTE_PUT_INHIBITED), causing the messages to be rolled back onto the original local instance of the queue and to wait there until a local application can be started to process them. If a message consumer is subsequently started on a remote queue manager, the redistribution process is not restarted, and therefore messages will remain queued even though there is a healthy message consumer application available elsewhere in the cluster. An application could be written to take the state of all queue instances into consideration before redistributing messages, and run at scheduled intervals until redistribution is successful.

If you use cluster workload rank, ensure that redistribution of messages to equally low-ranked queues does not result in messages cycling from queue manager to queue manager as part of a redistribution loop.

Message processing times are dictated by service level agreements. These times should be considered when setting the sampling interval for situations defined in Tivoli Enterprise Portal. WebSphere MQ triggering is usually the recommended alternative to starting applications based on monitoring and sampling, though as discussed, it is not suitable for all environments.

Although cluster auto-definition and object change propagation features are very useful, you need to ensure that channels between queue managers (especially to and from full repository queue managers) remain healthy. If channels are not healthy, cluster object change messages cannot flow, causing the workload balancing choice to be based on out-of-date data (for example, the put attribute of a queue). You can avoid this situation by monitoring channel status and alerting operations staff to channel problems, using products such as Omegamon XE for Messaging and Tivoli Monitoring.

### Conclusion

This article has shown you how to improve service availability in messaging systems by reducing failures and service timeouts caused by application outages, using WebSphere MQ clustering,
Omegamon XE for Messaging, and Tivoli Monitoring. These products improve availability by providing proven tools to manage and monitor your messaging infrastructure. These products also link well with a wide range of other Tivoli products to support a centralized, holistic approach to monitoring messaging systems.
Related topics

- **Omegamon XE for Messaging product page**
  Product descriptions, documentation, training information, support information, and more.

- **Omegamon XE for Messaging information center**
  A single Web portal to all Omegamon XE for Messaging documentation, with conceptual, task, and reference information on installing, configuring, and using Omegamon XE.

- **WebSphere MQ MA01 SupportPac**
  This SupportPac contains a simple pipeline program called Q that takes messages from one source and outputs them to a target. The source and target can be either the console, a queue, or a topic.

- **Cluster workload balancing algorithm**
  This topic from the WebSphere MQ V7 information center describes the workload management algorithm used when determining the final destination for messages being put onto cluster queues.

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

- **WebSphere MQ V7 information center**
  A single Web portal to all WebSphere MQ V7 documentation, with conceptual, task, and reference information on installing, configuring, and using WebSphere MQ V7.

- **IBM Redbook: WebSphere MQ V7 features and enhancements**
  Describes the fundamental concepts and benefits of message queuing technology, describes the new features in V7, and provides a business scenario that shows those features in action.

- **Download a free trial version of WebSphere MQ V7**
  A 90-day, full featured, no-charge trial of WebSphere MQ V7

- **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.

- **Tivoli Monitoring product page**
  Product descriptions, documentation, trials and demos, training information, support information, and more.

- **Tivoli Monitoring information center**
A single Web portal to information on deploying, installing, and configuring the common services components and the monitoring agents that make up the base Tivoli Monitoring product.

- **developerWorks WebSphere application connectivity developer resources**
  How-to articles, downloads, tutorials, education, product info, and other resources to help you build WebSphere application connectivity and business integration solutions.

- **developerWorks WebSphere business process management developer resources**
  WebSphere BPM how-to articles, downloads, tutorials, education, product info, and other resources to help you model, assemble, deploy, and manage business processes.

- **developerWorks WebSphere SOA and Web services developer resources**
  How-to articles, downloads, tutorials, education, product info, and other resources to help you design and build WebSphere SOA and Web services solutions.

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

- **WebSphere on-demand demos**
  Download, watch, and learn what WebSphere products and WebSphere-related technologies can do for your company.

- **developerWorks WebSphere weekly newsletter**
  The developerWorks newsletter gives you the latest articles and information only on those topics that interest you. In addition to WebSphere, you can select from Java, Linux, Open source, Rational, SOA, Web services, and other topics. Subscribe now and design your custom mailing.

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

- **developerWorks on Twitter**
  Check out recent Twitter messages and URLs.