Integrating an Oracle Service Bus Cluster with an IBM WebSphere MQ Cluster in a Service-Oriented Architecture

An Oracle White Paper
Updated January 2009

Integrating an Oracle Service Bus Cluster with an IBM WebSphere MQ Cluster in a Service-Oriented Architecture

Integrating an Oracle Service Bus Cluster with an IBM WebSphere MQ Cluster in a Service-Oriented Architecture

INTRODUCTION

Service-oriented architecture has become a widely accepted industry paradigm driven in large part by the increased demand for the asynchronous exchange of business data. However, messaging middleware can vary across organizations, which means that Oracle must provide a way to integrate its product families with each of these middleware solutions. In particular, customers should be able to use the integrated Oracle Service Bus/IBM WebSphere MQ system to provide support for messaging interoperability, load balancing, high availability, high performance, failover, and exactly once quality-of-messaging service (with regard to request messages).

This white paper discusses the setup and configuration of a clustered Oracle Service Bus and WebSphere MQ with the WebSphere MQ Extended Transactional Client. It focuses on the distributed transactional request/response messaging communication between these products.

This white paper discusses the setup and configuration of a clustered Oracle Service Bus (release 2.1 and later) and WebSphere MQ (release 5.3 and later) with the IBM WebSphere MQ Extended Transactional Client. It focuses on the distributed transactional request/response messaging communication between these products. The example provided allows customers to achieve load balancing and failover in an Oracle Service Bus cluster domain, as well as load balancing in a WebSphere MQ cluster.

SELECTING A MESSAGE INFRASTRUCTURE

Java Message Service (JMS) in Oracle Service Bus is Oracle WebLogic implementation of JMS 1.1 specification. It is an enterprise-class messaging system that not only fully supports the JMS 1.1 specification, but also provides numerous extensions to the standard JMS APIs. Because JMS functionality in Oracle Service Bus is tightly integrated into the Oracle WebLogic Server platform, customers can build highly secure Java enterprise applications that can be monitored through the Oracle Service Bus administration console. In addition to fully supporting extended architecture (XA) transactions, JMS in Oracle Service Bus offers high availability through its clustering and server migration features. It also provides seamless interoperability with third-party messaging vendors.

When deciding to implement an integrated system, it is worth weighing the trade-offs between using an Oracle Service Bus/WebSphere MQ messaging conglomerate and using an Oracle Service Bus/Oracle WebLogic JMS. Usually the main reason for using WebSphere MQ with Oracle Service Bus is to support a legacy messaging system that is already in place and not easily replaceable.

Usually the main reason for using WebSphere MQ with Oracle Service Bus is to support a legacy messaging system that is already in place and not easily replaceable. Customers might instead want to use the Oracle Service Bus/Oracle Weblogic JMS, because it is easy to use, configure, and monitor, and it provides performance and features that meet or exceed those of other JMS vendors.

Customers might instead want to use the Oracle Service Bus/Oracle WebLogic JMS, because it is easy to use, configure, and monitor, and it provides performance and features that meet or exceed those of other JMS vendors. Using Oracle Service Bus/Oracle WebLogic JMS offers the following benefits:

- It runs as an integral part of Oracle WebLogic Server, whereas a foreign JMS provider must be configured, started, stopped, and monitored using a separate set of tools.
- It is integrated with the built-in Java Naming and Directory Interface (JNDI) and clustering support of Oracle WebLogic Server, so there is no need to configure and manage a separate JNDI infrastructure, whether it is a collection of files or a Lightweight Directory Access Protocol (LDAP) server.
- It provides strict conformance with the JMS specification, so that applications can be written in a portable way.
- It provides excellent performance and scalability.
- It provides more-efficient transaction management for JMS. When a foreign JMS provider is used, both Oracle Service Bus and the foreign provider must undergo the extra overhead of two-phase commit between Oracle Service Bus and the JMS provider.

Integrating Oracle Service Bus and WebSphere MQ

The goal of using an integrated system is to connect the WebSphere MQ messaging system to Oracle Service Bus using the WebSphere MQ JMS interface. The user will then typically use the WebSphere MQ native protocol for business processes. Distributed transactional integrity, reliability, scalability, and failover need to be ensured in the integrated system, and the use of clustered Oracle Service Bus and WebSphere MQ architectures should be optimized.

DESIGN AND CONFIGURATION

The following sections of this white paper outline the features directly related to Oracle Service Bus/WebSphere MQ intercluster communication.[1]

[1] For more information on Oracle WebLogic Server clusters and IBM WebSphere MQ queue manager clusters, see the Appendix.

Distributed Transactional Integrity

For WebSphere MQ release 5.3 and later, support is offered for XA transaction management in conjunction with an external syncpoint coordinator. This support is available in both the JMS and C clients. The WebSphere MQ Extended Transactional Client lets applications participate within a unit of work with other local resource managers, under the control of an external syncpoint coordinator. Until the release of the WebSphere MQ Extended Transactional Client in February 2003, this was only possible by using a local MQ server. For the purposes of pricing, the WebSphere MQ Extended Transactional Client is treated as if it were an MQ server while the current nontransactional client continues to be provided free of charge.

When a WebSphere MQ Extended Transactional Client is colocated with each Oracle Service Bus cluster server, distributed transactional messaging is possible between Oracle Service Bus and remote WebSphere MQ servers.

When a WebSphere MQ Extended Transactional Client is colocated with each Oracle Service Bus cluster server, distributed transactional messaging is possible between Oracle Service Bus and remote WebSphere MQ servers.

Terminology	Explanation
Resource manager	A computer subsystem that owns and manages resources that can be accessed and updated by applications (IBM WebSphere MQ queue manager—queues are its resources; IBM DB2 database—tables are its resources)
Unit of work	When an application updates the resources of one or more resource managers, it is often vital that all updates complete successfully as a group, or none of them will complete. Updates that complete this way are said to occur within a "unit of work" or "transaction."
Syncpoint coordinator	A syncpoint is the point in time when all updates within a unit of work are either committed or backed out. The computer subsystem that manages units of work is called a syncpoint coordinator.
Extended architecture (XA)	For a transaction manager to manage a unit of work, an architected interface to the resource manager must be present. One such interface is the XA interface, which is published by X/Open Company Limited (now the Open Group).

Oracle Service Bus Distributed Transactional Support

When Java transactional APIs are distributed via Oracle WebLogic Server, transparent request/response messaging is enabled between an Oracle Service Bus JMS cluster and a WebSphere MQ cluster.

WebSphere MQ Cluster Setting

If a WebSphere MQ cluster node that serves as a remote message forwarder goes down and is restarted, it is possible to keep request/response messaging interoperability uninterrupted and transparent between an Oracle Service Bus JMS cluster and a WebSphere MQ cluster, by using a clustered MQ setting. This is because the queue manager that holds the local references to shared queues stays alive. Oracle Service Bus can still put messages to the request queue and get messages from the response queue. A business application could still get messages from the request queue and put messages to the response queue.

However, this is not the case if the queue manager that holds the local references to shared request and response queues goes down. In that case, messages can still be put to the queues, but not retrieved from them.

TRANSPARENT REQUEST/RESPONSE MESSAGING SETUP AND CONFIGURATION

In general, the details of the installation and configuration depend on the operating system, the number of machines, and the desired cluster sizes of Oracle Service Bus and WebSphere MQ. In a production environment, each Oracle Service Bus cluster server is usually hosted on a separate machine. In addition, each WebSphere MQ cluster node can be hosted on a separate hardware system.

The following section offers an example setup and configuration for transparent request/response messaging between an Oracle Service Bus JMS cluster and a WebSphere MQ cluster. Minimal hardware is used—two systems, each running Microsoft Windows operating systems.

The following section offers an example setup and configuration for transparent request/response messaging between an Oracle Service Bus JMS cluster and a WebSphere MQ cluster. Minimal hardware is used—two systems, each running Microsoft Windows operating systems. The first machine hosts the Oracle Service Bus domain and the WebSphere MQ Extended Transactional Client. The second machine hosts a cluster of WebSphere MQ servers.

The Oracle Service Bus domain includes an administration server and a cluster of two managed servers. The WebSphere MQ cluster includes two nodes. The example below outlines common configuration steps that can be extrapolated to suit production requirements.

Sample Installation of a WebSphere MQ Server and a WebSphere MQ Extended Transactional Client

To install and configure a WebSphere MQ server and a WebSphere MQ Extended Transactional Client, begin by installing WebSphere MQ 5.3 or 6.0. Follow the step-by-step installation instructions at the time of installation, and, if needed, consult the IBM WebSphere MQ information center cited in the Appendix.

Next, create two (or more) queue managers in WebSphere MQ. Setting queue managers as cluster members offers advantages over distributed queuing. When clustering is not used, the queue managers are independent and communicate using distributed queuing. In that case, if one queue manager needs to send messages to another, it must define both a transmission queue and a channel to the remote queue manager.

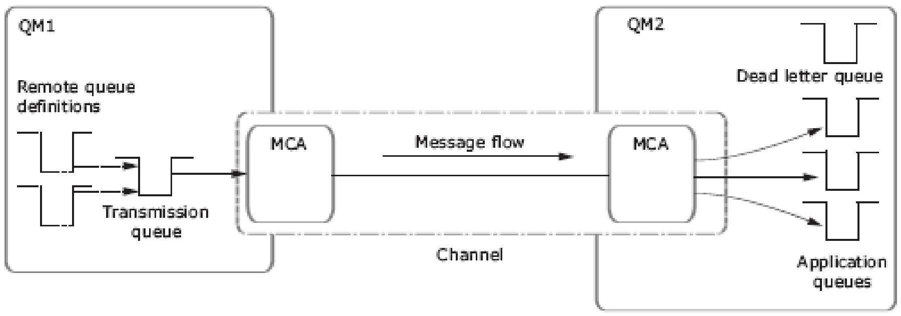

Figure 1: The components required for distributed queuing

Any queue manager can send a message to any other queue manager in the same cluster without explicit channel definitions, remote queue definitions, or transmission queues for each destination.

However, if queue managers are grouped in a cluster, the queue managers can make the queues that they host available to every other queue manager in the cluster. Any queue manager can send a message to any other queue manager in the same cluster without explicit channel definitions, remote queue definitions, or transmission queues for each destination. Every queue manager in a cluster has a single transmission queue from which it can transmit messages to any other queue manager in the cluster. Each queue manager in a cluster only needs to define one cluster-receiver channel on which to receive messages, and one cluster-sender channel with which it introduces itself and learns about the cluster. Figure 2 shows a small cluster of queue managers.

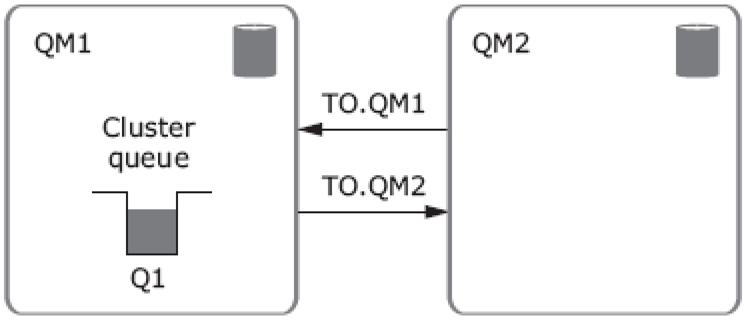

Figure 2: A small cluster of two queue managers

Next, create a cluster and include the queue managers as cluster nodes. Then specify both queue managers as hosting repositories.

The full repository queue managers, QM1 and QM2, host repositories of information about the queue managers in the cluster. (The repositories are represented in Figure 2 by the shaded cylinders.) QM1 hosts some queues that are accessible to any other queue manager in the cluster. These are called cluster queues. (The cluster queues are represented in Figure 2 by the shaded queues.)

As with distributed queuing, applications use the MQPUT call to put a message on a cluster queue at any queue manager. Applications use the MQGET call to retrieve messages from a cluster queue on the local queue manager.

Including queue managers in the cluster involves creating system channels between them. Each queue manager has a definition for the receiving end of a channel called TO.qmgr on which it can receive messages. This is a cluster-receiver channel. A cluster-receiver channel is similar to a receiver channel used in distributed queuing, but in addition to carrying messages, this channel can also carry information about the cluster.

Each queue manager also has a definition for the sending end of a channel, which connects to the cluster-receiver channel of one of the full repository queue managers. This is a cluster-sender channel. In Figure 2, QM1 and QM2 have cluster-sender channels that connect to TO.QM2. QM2 has a cluster-sender channel that connects to TO.QM1. A cluster-sender channel is similar to a sender channel used in distributed queuing, but in addition to carrying messages, this channel can also carry information about the cluster.

Once both the cluster-receiver end and the cluster-sender end of a channel have been defined, the channel starts automatically.

Once both the cluster-receiver end and the cluster-sender end of a channel have been defined, the channel starts automatically.

On Windows systems, creating queue managers and a cluster, and including queue managers in the cluster, can be done from the WebSphere MQ Explorer interface or by using the command-line interface (CLI) tool (runmqsc) found in the …/WebSphere MQ/bin directory. On UNIX systems, use the CLI tool runmqsc.

Example for the Windows operating system:

```
>runmqsc.exe QM_IR02.1416_ALSB1
ALTER QMGR REPOS(ALSB1)

DEFINE CHANNEL(TO.QM_IR02.1416_ALSB1) CHLTYPE(CLUSRCVR)
TRPTYPE(TCP) CONNAME(‘ir02.bea.com(1416)') CLUSTER(ALSB1)

DEFINE CHANNEL(TO.QM_IR02.1417_ALSB1) CHLTYPE(CLUSSDR)
TRPTYPE(TCP) CONNAME(‘ir02.bea.com(1417)') CLUSTER(ALSB1)

>runmqsc.exe QM_IR02.1417_ALSB1
ALTER QMGR REPOS(ALSB1)

DEFINE CHANNEL(TO.QM_IR02.1417_ALSB1) CHLTYPE(CLUSRCVR)
TRPTYPE(TCP) CONNAME(‘ir02.bea.com(1417)') CLUSTER(ALSB1)

DEFINE CHANNEL(TO.QM_IR02.1417_ALSB1) CHLTYPE(CLUSSDR)
TRPTYPE(TCP) CONNAME(‘ir02.bea.com(1416)') CLUSTER(ALSB1)
```

The next step is to create two local queues (request and response) on one of the queue managers using the WebSphere MQ Explorer interface or runmqsc, and share them on the cluster.

Example:

```
DEFINE QLOCAL(REQUESTQUEUE1) CLUSTER(ALSB1)

DEFINE QLOCAL(RESPONSEQUEUE1) CLUSTER(ALSB1)
```

The WebSphere MQ Extended Transactional Client 5.3 comes as a separate installation package for release 5.3. It enables global transactions between Oracle WebLogic Server and WebSphere MQ hosted on separate machines.

Next, install the WebSphere MQ Extended Transactional Client 5.3 on a second machine. The WebSphere MQ Extended Transactional Client 5.3 comes as a separate installation package for release 5.3. It enables global transactions between Oracle WebLogic Server and WebSphere MQ hosted on separate machines. The Appendix section of this document points to a complete explanation.

Then install the Oracle Service Bus cluster domain on the same machine where the WebSphere MQ Extended Transactional Client 5.3 is installed.

The above setup is somewhat simplified, because in production each managed server will be on a separate machine and has to be colocated with its own installation of the WebSphere MQ Extended Transactional Client.

Configuring Java Message Service—Foreign Java Message Service Providers

To explain the configuration of JMS using JNDI, this white paper uses information from the "Using Foreign JMS Providers with WebLogic Server" document, which can be found at http://ftpna2.bea.com/pub/downloads/jmsproviders.pdf.

A JMS client needs to look up an initial connection factory object and destination objects—queues and topics—using JNDI. Therefore, a JMS client can use a WebSphere MQ JMS provider when two key settings are implemented:

- JNDI binding of the WebSphere MQ JMS objects using WebSphere MQ command-line utility. Besides connection factories and destinations, the WebSphere MQ JMS provider uses JNDI to store its own configuration information, such as the IP address and port of the server, type of transport, destination attributes, and other special options.
- Configuration of the foreign JMS provider for WebSphere MQ using the Oracle WebLogic Server administration console.

This configuration supplies the following information to the Oracle Service Bus server:

- **Initial context factory.** This is the name of the JNDI class that will be created to perform the lookup. The JNDI class can be part of the Java Development Kit, it can be provided by Sun Microsystems, or it can be provided by the JMS vendor.
- **Provider URL.** This URL tells the initial context factory where to find the directory information. It can refer to a server using a protocol such as LDAP, it can refer to a file in the file system, or it can refer to something else.

- **Connection factory JNDI name.** This is the name of the connection factory object that is stored in JNDI. Each JMS provider creates its own tool to create this object and store it in JNDI. Once it is stored there, any JMS client program (including Oracle Service Bus and Oracle WebLogic Server) can look up the object.
- **Destination JNDI name.** This object represents a JMS destination, a queue, or a topic, as stored in JNDI. Like the connection factory, each JMS provider will provide its own tool to create the object and place it in JNDI.

Configuring Java Message Service—WebSphere MQ Java Message Service

To configure WebSphere MQ JMS, perform identical JNDI binding of the WebSphere MQ JMS objects on both machines—with WebSphere MQ server and the WebSphere MQ Extended Transactional Client installations. Alternatively, the shared data structure can be used.

First, select the JNDI service provider and the URL of the service provider's initial context by uncommenting them in the following file: …/Java/bin/JMSAdmin.config.

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
PROVIDER_URL=file:/C:/JNDI-Directory
```

Second, bind JMS objects into the JNDI tree using the JMS admin file …/Java/bin/JMSAdmin.bat. The bindings will be stored in the file C:\JNDI-Directory\.binding. Namely:

- Bind two WebSphere MQ XA connection factories (one for each WebSphere MQ server node) indicating the name of the factory, the name of the queue manager, host name, port, and client transport. For example:

```
def xaqcf(XAQCF0) qmanager(QM_IR02.1416_ALSB1) HOSTNAME(ir02)
PORT(1416) TRANSPORT(CLIENT)

def xaqcf(XAQCF1) qmanager(QM_IR02.1417_ALSB1) HOSTNAME(ir02)
PORT(1417) TRANSPORT(CLIENT)
```

- Bind two MQ queues (request and response) indicating the name of the queue and the name of the queue manager holding the local reference to the queue and persistence mode. If the client that dequeues the queue will be a native WebSphere MQ client, the TARGCLIENT(MQ) option must be specified. This option ensures that the client will be able to retrieve proper WebSphere MQ headers for the message. For example:

```
def Q(REQUESTQUEUE1) queue(REQUESTQUEUE1)
qmanager(QM_IR02.1416_ALSB1) PERSISTENCE(PERS) TARGCLIENT(MQ)

def Q(RESPONSEQUEUE1) queue(RESPONSEQUEUE1)
qmanager(QM_IR02.1416_ALSB1) PERSISTENCE(PERS)
```

Finally, to see the resulting WebSphere MQ JMS bindings, give the command dis ctx:

```
InitCtx> dis ctx

   Contents of InitCtx

         .bindings                              java.io.File
     a    RESPONSEQUEUE1                   com.ibm.mq.jms.MQQueue
     a    REQUESTQUEUE1                    com.ibm.mq.jms.MQQueue
     a    XAQCF0
 com.ibm.mq.jms.MQXAQueueConnectionFactory



     a    XAQCF1
 com.ibm.mq.jms.MQXAQueueConnectionFactory
     5 Object(s)
     0 Context(s)
     5 Binding(s), 4 Administered
```

Configuring Java Message Service—Oracle Service Bus Java Message Service

The simplest setting is to configure the Oracle Service Bus cluster domain in production mode with an Oracle database and two colocated managed servers.

Edit setDomainEnv.cmd or setDomainEnv.sh.
ADD EXTENSION TO PRE_CLASSPATH.

For example:

```
set IBM_MQ_DIR=C:\Program Files\IBM\WebSphere MQ\Java\lib
set IBM_MQ_ETC_DIR=C:\Program
Files\IBM\Source\C48UVML\Windows\MSI\Program Files\IBM\WebSphere
MQ\Java\lib
set
EXT_PRE_CLASSPATH=%IBM_MQ_DIR%\com.ibm.mqjms.jar;%IBM_MQ_DIR%\fscontext.jar;
%IBM_MQ_DIR%\com.ibm.mq.jar;%IBM_MQ_DIR%\com.ibm.mqbind.
jar;%IBM_MQ_DIR%\providerutil.jar;%IBM_MQ_ETC_DIR%\com.ibm.mqetclient.jar;
%EXT_PRE_CLASSPATH%
```

Optionally, for demonstration purposes, to reduce the memory consumption, change the default memory settings.

For example:

```
set MEM_ARGS=-Xms256m -Xmx256m
```

Start the administration and clustered (managed) servers.

To have all relevant JMS objects in a separate JMS system module, use the Oracle WebLogic Server administration console to create a new JMS system module that is deployed on the cluster.

In this JMS system module, create

- A foreign JMS server deployed on the cluster with two WebSphere MQ XA connection factories corresponding to two WebSphere MQ cluster nodes—XAQCF0 and XAQCF1—and two queues, the WebSphere MQ request and response queues REQUESTQUEUE1 and RESPONSEQUEUE1
- An XA connection factory with default delivery mode set to PERSISTENT
- A proxy request Gateway Uniform Distributed Queue (UDQ) deployed on the cluster
- A proxy response UDQ deployed on the cluster

The purpose behind creating the local queue on which the first proxy service will be deployed is to route the message to the second proxy service deployed on the UDQ. This is purely optional, but can be done to see all cluster servers that participate in sending messages to the WebSphere MQ server.

Optionally, if using a Plain Old Java Object (POJO) JMS client to send request messages for testing the system, create a local queue that is deployed on one of the managed servers as an entry point. The purpose behind creating the local queue on which the first proxy service will be deployed is to route the message to the second proxy service deployed on the UDQ. This is purely optional, but can be done to see all cluster servers that participate in sending messages to the WebSphere MQ server.

Next, verify that the JNDI panel in the Oracle WebLogic Server administration console shows the JNDI tree with all foreign JMS server objects (WebSphere MQ connection factories and queues) and the newly created Oracle Service Bus JMS connection factory and queues.

Using the Oracle Service Bus console (sbconsole), carry out the following actions:

- Configure the first JMS proxy service with the local queue as JMS endpoint (MDB will be deployed on that local queue). This is optional if using POJO JMS client to send the initial message.
- Configure the second JMS proxy service with the Gateway UDQ as JMS endpoint (MDB deployed on the Gateway UDQ). This JMS proxy service requires a response to the response UDQ.
- Configure the router from the first JMS proxy service to the second one.

- Configure the JMS business service deployed on the WebSphere MQ request queue that requires response to the WebSphere MQ response queue. This business service has two endpoints, each pointing to one of the two WebSphere MQ queue managers for WebSphere MQ cluster load balancing.
- Configure the pipeline between the second JMS proxy service and the JMS business service with the router that will route messages from the second JMS proxy service to the JMS business service.
- Configure the Request and Response Logging Actions for logging messaging headers. The logging will show how many request/response messages would be passing the request/response pipelines.

Figure 3 shows an example of an Oracle Service Bus/WebSphere MQ setup.

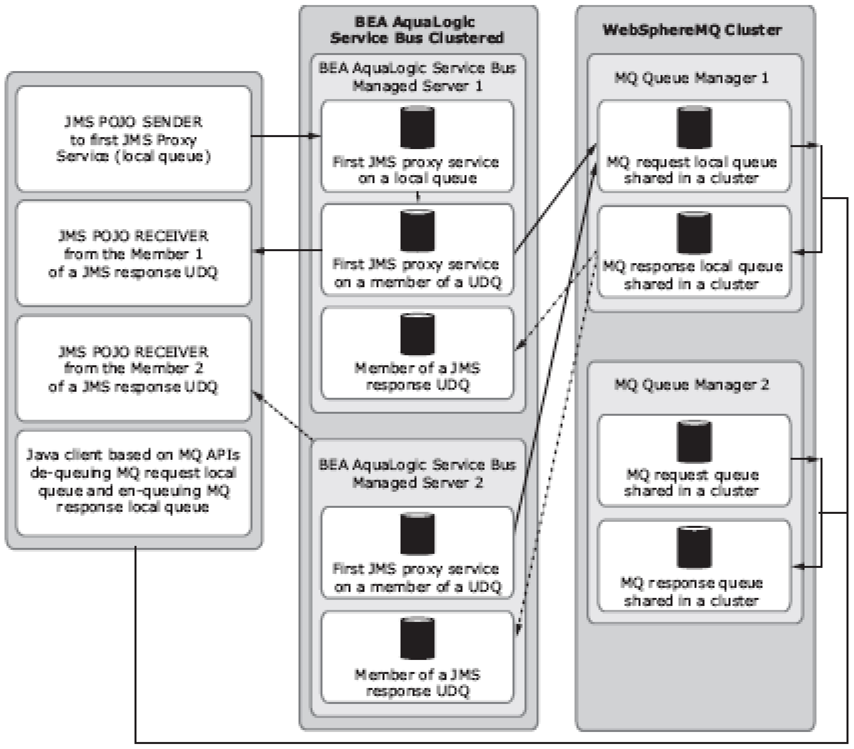
BEA AquaLogic Service Bus Clustered
WebSphereMQ Cluster
BEA AquaLogic Service Bus Managed Server 1
MQ Queue Manager 1
JMS POJO SENDER to first JMS Proxy Service (local queue)
First JMS proxy service on a local queue
MQ request local queue shared in a cluster
JMS POJO RECEIVER from the Member 1 of a JMS response UDQ
First JMS proxy service on a member of a UDQ
MQ response local queue shared in a cluster
Member of a JMS response UDQ
JMS POJO RECEIVER from the Member 2 of a JMS response UDQ
MQ Queue Manager 2
Java client based on MQ APIs de-queuing MQ request local queue and en-queuing MQ response local queue
BEA AquaLogic Service Bus Managed Server 2
MQ request queue shared in a cluster
First JMS proxy service on a member of a UDQ
MQ response queue shared in a cluster
Member of a JMS response UDQ

Figure 3: Test setup—request/response messaging communication between an Oracle Service Bus cluster domain and a WebSphere MQ cluster

TESTING THE SYSTEM

The next step is to test the newly integrated system. Verifying the messaging between the Oracle Service Bus and WebSphere MQ clusters with Oracle Service Bus XA transactional managers and the WebSphere MQ XA resource manager requires the following:

Verifying the messaging between the Oracle Service Bus and WebSphere MQ clusters with Oracle Service Bus XA transactional managers and the WebSphere MQ XA resource manager requires some configuration.

- XA connection factories in both WebSphere MQ and Oracle Service Bus
- A PERSISTENT delivery mode
- A JMS client that sends a request to the Oracle Service Bus proxy service
- A business service based on WebSphere MQ APIs, which receives the request message and sends a response message
- A JMS client that receives the response message

The business service can be written as a class, based on WebSphere MQ Java APIs; the service gets the request message from the request queue and puts the response message in the response queue. If the WebSphere MQ class is hosted remotely to the WebSphere MQ server, a server connection channel and a corresponding client channel must be created on the WebSphere MQ server to which the WebSphere MQ client connects.

The WebSphere MQ class takes care of setting environment variables, initializing WebSphere MQ resources, measuring the current depth of the request queue, looping through, and getting the messages from the request queue. It sets the correlation ID on each response message to the request correlation ID, puts the response messages in the response queue, and releases the WebSphere MQ resources.

If and when the common messaging request/response pattern is supported in the Oracle Service Bus JMS request/response implementation, the WebSphere MQ class will set the correlation ID on the response message to the request message ID.

It is possible to write POJO JMS clients. One POJO JMS client will send the request (for example, an XML file) to the first proxy service. Each of the two additional POJO JMS clients will listen to one of the two members of the proxy service's response UDQ.

These are the settings:

- The client sends request messages to WebSphere MQ via Oracle Service Bus. It uses the standard JMS API protocol or enhanced WebLogic JMS APIs. An Oracle Service Bus proxy service receives the client's messages and dispatches them to the business service deployed on WebSphere MQ.
- Oracle Service Bus uses the underlying Oracle WebLogic JMS—an enterprise-class messaging system that supports the JMS 1.1 specification and also provides important and useful extensions of the standard JMS APIs.
- The business service uses native WebSphere MQ as the transport protocol.
- A WebSphere MQ cluster queue was created by creating a local queue on one of the queue managers and sharing it with a second cluster member. The second cluster member will be a remote forwarder to the shared queue. WebSphere MQ is essentially a store-and-forward message-passing middleware protocol.

- Requests from Oracle Service Bus can be load balanced by sending messages in a round-robin fashion to the members of the cluster. However, getting the response message (dequeue the response queue that is shared on a cluster) is only possible by accessing the WebSphere MQ server node that holds the local reference to the queue. In other words, when sending a request message, Oracle Service Bus can iterate through a list of the JMS destinations. But it can receive a response message only from one JMS destination.

Test Scenario 1: Managed Servers Alive and Connected

First, test the scenario in which all the managed servers in the Oracle Service Bus cluster are alive and connected, and all WebSphere MQ cluster servers are also alive and connected.

Messages can be sent directly to the second Oracle Service Bus JMS proxy service deployed on the Gateway UDQ, if the sender is capable of distributing the messages between the members of the distributed queue.

Messages can be sent directly to the second Oracle Service Bus JMS proxy service deployed on the Gateway UDQ, if the sender is capable of distributing the messages between members of the distributed queue. Otherwise, use a POJO JMS client to send the predefined amount of request messages (perhaps reading them from XML files) to the first JMS proxy service.

- The first JMS proxy service routes the request message to the second JMS proxy service.
- The second JMS proxy service is deployed on the Gateway UDQ and requires response to the response UDQ.
- The second JMS proxy service routes the request message to the JMS business service.
- The JMS business service is deployed on the WebSphere MQ request queue and requires response to the WebSphere MQ response queue.
- The JMS business service has two endpoints, each on one of the two WebSphere MQ queue managers. The round-robin algorithm is selected for the load balancing between these two endpoints.
- A Java client based on the WebSphere MQ APIs measures the current depth of the request queue and loops through the messages to get the request message from the request queue and puts the response message in the response queue. As required for Oracle Service Bus 2.1, it reads the request correlation ID as byte[ ] and sets it as byte[ ] correlation ID on the response message.
- To find out whether a traditional enterprise application integration messaging request/response pattern is supported, consult the user documentation for Oracle Service Bus 2.5. If so, in Oracle Service Bus 2.5, the business service application should read request message ID as byte[ ]. Set it as byte[ ] correlation ID on the response message.

- A response MDB listens on the WebSphere MQ response queue, processes response messages, and puts them in the response pipeline to be delivered to the response UDQ.
- Two POJO JMS clients listen, each to one of the two members of the response UDQ. They receive messages, read their correlation ID, and log it along with the current count of the received messages.

Test Scenario 2: Oracle Servers Alive, One WebSphere MQ Server Restarted

The Java client based on MQ APIs measures the current depth of the request queue and loops through the messages to move them from the request queue to the response queue.

Second, test the scenario in which all managed servers in the Oracle Service Bus cluster are alive, but one of the WebSphere MQ cluster servers is stopped and restarted. To implement it, carry out these instructions:

- Start the Oracle Service Bus administration server and two managed servers.
- Stop one of the WebSphere MQ queue managers of the cluster (on which local request/response queues were configured).
- Send a predefined number of messages (XML files) to the first JMS proxy service using a POJO JMS client.
- Start the WebSphere MQ queue manager that was stopped.
- Observe that all messages sent to WebSphere MQ were successfully delivered to the WebSphere MQ request queue—through the second WebSphere MQ cluster node that stayed alive.
- The Java client based on WebSphere MQ APIs measures the current depth of the request queue and loops through the messages to move them from the request queue to the response queue. It reads the request correlation ID as byte and sets it as byte[] correlation ID on the response message.
- A response MDB listens on the WebSphere MQ response queue, processes response messages, and puts them in the response pipeline to be delivered to the proxy service response UDQ.
- Two POJO JMS clients listen to one of the two members of the proxy service response UDQ. They receive messages, read their correlation ID, and log it and the current count of the received messages.

Test Scenario 3: Oracle Servers Alive, Entire WebSphere MQ Cluster Restarted

Third, test the scenario in which all managed servers in the Oracle Service Bus cluster are alive, but the WebSphere MQ cluster is stopped and restarted. To implement it, carry out these instructions:

- Start the Oracle Service Bus administration server and two managed servers.
- Stop WebSphere MQ services on the remote machine. Expect to receive repeated disconnect-related exceptions on both managed servers.

- Send a predefined number of messages (XML files) to the first JMS proxy service using a POJO JMS client.
- Start the WebSphere MQ cluster.
- Observe that all messages that were sent to WebSphere MQ were successfully delivered to the WebSphere MQ request queue by Oracle Service Bus.

CONCLUSION

This white paper describes an example of the configuration of the messaging communication between Oracle Service Bus and WebSphere MQ queue manager clusters. It is an initial attempt to find the best way of configuring such a system. Descriptions of real-life installations of WebSphere MQ clusters from field engineers would be much appreciated (so they can be reproduced at Oracle). If those configurations are significantly different from the type of WebSphere MQ cluster described in this paper, the additional information can be appended to the white paper.

Starting from Oracle Service Bus 2.1, users can configure request/response queues that are shared on the WebSphere MQ cluster and separate XA connection factories for each WebSphere MQ server, and send messages to those WebSphere MQ servers in a round-robin manner. In this way, WebSphere MQ load balancing is driven by Oracle Service Bus.

Users can verify that the request messages are delivered to different servers. When a message arrives to the WebSphere MQ server that holds the local reference at the cluster request queue, it stays there. When a message arrives at the remote forwarder, the remote forwarder immediately forwards it to the server that holds the local reference to the cluster request queue. Browsing that cluster queue from the local queue manager shows all messages that were sent to the cluster.

A JMS service in Oracle Service Bus can get a response message from only one destination—the cluster response queue on its local queue manager.

In a request/response XA transactional clustered Oracle Service Bus/clustered WebSphere MQ scenario, messages are not lost when one of the WebSphere MQ cluster servers is shut down and restarted.

In a request/response XA transactional clustered Oracle Service Bus/clustered WebSphere MQ scenario, messages are not lost when one of the WebSphere MQ cluster servers is shut down and restarted. In fact, if the server that is shut down is a remote forwarder, operations continue smoothly because messages continue to arrive to the local queue manager of the cluster request queue. In addition, the response messages are retrieved from its cluster response queue. The messages are not lost when the entire WebSphere MQ cluster is shut down and restarted.

APPENDIX: REFERENCES AND RELATED DOCUMENTS

- **Messaging (JMS) for Oracle WebLogic Server 10.3**
  e-docs.bea.com/wls/docs103/messaging.html
- **Configure Foreign Servers**
  e-docs.bea.com/wls/docs103/ConsoleHelp/taskhelp/jms_modules/
  foreign_servers/ConfigureForeignServers.html
- **Accessing Foreign Server Providers**
  e-docs.bea.com/wls/docs103/jms_admin/advance_config.html#1075917
- **FAQ: Integrating Remote JMS Providers**
  e-docs.bea.com/wls/docs81/faq/interop.html
- **Using Foreign JMS Providers with Oracle WebLogic Server**
  ftpna2.bea.com/pub/downloads/jmsproviders.pdf
- **Using Oracle WebLogic Server Clusters**
  e-docs.bea.com/wls/docs103/cluster/index.html
- **Communications in a Cluster**
  e-docs.bea.com/wls/docs9103/cluster/features.html
- **IBM WebSphere MQ Information Center**
  publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp
- **WebSphere MQ: Information Roadmap for All Users**
  www-128.ibm.com/developerworks/websphere/zones/businessintegration/
  roadmaps/wmq/
- **WebSphere MQ Queue Manager Clusters**
  publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp
- **IBM WebSphere MQ Extended Transactional Client**
  www-306.ibm.com/software/integration/wmq/transclient.html
- **Meet the Experts: Mark Taylor on WebSphere MQ**
  www-128.ibm.com/developerworks/websphere/library/techarticles/
  0302_taylor/taylor.html

Integrating an Oracle Service Bus Cluster Domain with an
IBM WebSphere MQ Cluster in a Service-Oriented Architecture
Updated January 2009

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com