IBM WebSphere Application Server for z/OS, Version 8.0

Establishing highly available services for applications

SA38-0664-01

Note

Before using this information, be sure to read the general information under “Notices” on page 41.

Compilation date: July 14, 2011

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-5250.

  Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Changes to serve you more quickly

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Under construction!

The Information Development Team for IBM WebSphere Application Server is changing its PDF book delivery strategy to respond better to user needs. The intention is to deliver the content to you in PDF format more frequently. During a temporary transition phase, you might experience broken links. During the transition phase, expect the following link behavior:

- Links to Web addresses beginning with http:// work
- Links that refer to specific page numbers within the same PDF book work
- The remaining links will *not* work. You receive an error message when you click them

Thanks for your patience, in the short term, to facilitate the transition to more frequent PDF book updates.

Chapter 1. Establishing high availability for Data access resources

This page provides a starting point for finding information about data access. Various enterprise information systems (EIS) use different methods for storing data. These backend data stores might be relational databases, procedural transaction programs, or object-oriented databases.

The flexible IBM WebSphere® Application Server provides several options for accessing an information system's backend data store:

- Programming directly to the database through the JDBC 4.0 API, JDBC 3.0 API, or JDBC 2.0 optional package API.
- Programming to the procedural backend transaction through various J2EE Connector Architecture (JCA) 1.0 or 1.5 compliant connectors.
- Programming in the bean-managed persistence (BMP) bean or servlets indirectly accessing the backend store through either the JDBC API or JCA compliant connectors.
- Using container-managed persistence (CMP) beans.
- Using the IBM data access beans, which also use the JDBC API, but give you a rich set of features and function that hide much of the complexity associated with accessing relational databases.

Service Data Objects (SDO) simplify the programmer experience with a universal abstraction for messages and data, whether the programmer thinks of data in terms of XML documents or Java objects. For programmers, SDOs eliminate the complexity of the underlying data access technology (JDBC, RMI/IIOP, JAX-RPC, JMS, and so on) and message transport technology (java.io.Serializable, DOM Ojbects, SOAP, JMS, and so on).

Changing the error detection model to use the Exception Checking Model

The error detection model has been expanded and the data source has a configuration option that you can use to select the exception mapping model or the exception checking model for error detection. This configuration option allows the Error Detection Model to comply with Java Database Connectivity (JDBC) 4.0.

About this task

By default, the exception mapping Error Detection Model configuration is selected. The exception mapping Error Detection Model replaces some exceptions raised by the JDBC driver. Exception checking does not do this. If you want to use this configuration, no changes are needed. If you want to use the exception checking model, you need to configure the error detection model in the application server. If you previously changed the **Error Detection Model**, you can also use these steps to change the configuration back to using to the exception mapping model.

Procedure

1. Open the administrative console.
2. Go to the **WebSphere Application Server Data Source properties** panel for the data source.
   a. Select **Resources** > **JDBC** > **Data Sources** > ***data_source***
   b. Select **WebSphere Application Server Data Source properties**.
3. In the **Error Detection Model** section, click **Use the WebSphere Application Server Exception Checking Model**.

Configuring resource adapters

You can view a list of installed and configured resource adapters in the administrative console. Also, you can use the administrative console to install new resource adapters, create additional configurations of installed resource adapters, or delete resource adapter configurations.

Before you begin

A resource adapter is an implementation of the Java EE Connector Architecture (JCA) specification. The JCA specification provides access for applications to resources outside of the server or provides access for an enterprise information system (EIS) to applications on the server. It can provide application access to resources such as DB2®, Customer Information Control System (CICS®), Information Management Systems (IMS™), SAP, and PeopleSoft.

It can provide an EIS with the ability to communicate with message-driven beans that are configured on the server. Some resource adapters are provided by IBM®; however, third-party vendors can provide their own resource adapters. A resource adapter implementation is provided in a resource adapter archive (RAR)file; this file has an extension,`RAR`. A resource adapter can be provided as a stand-alone adapter or as part of an application, in which case it is called an embedded adapter.

The Java Connector Architecture (JCA) Version 1.6 specification adds support for Java annotations and Bean Validation in RAR modules. For more information about annotation support and metadata, see the topic, JCA 1.6 support for annotations in RAR modules.

About this task

Use this task to configure a stand-alone resource adapter archive file. Embedded adapters are installed as part of the application installation. This panel can be used to work with either type adapter.

Procedure

1. Open the product administrative console.
2. Select **Resources** > **Resource adapters** > *resource_adapter*.
3. Set the scope setting. This field specifies the level to which this resource definition is visible. For general information, see the topic, Administrative console scope settings, in the Related Reference section. The Scope field is a read-only string field that shows where the particular definition for a resource adapter is located. This field is set either when the resource adapter is installed, which can only be at the node level, or when a new resource adapter definition is added.
4. Configure the description. This field specifies a text description of the resource adapter. Use a free-form text string to describe the resource adapter and its purpose.
5. Set the archive path. Use this field to specify the path to the RAR file containing the module for this resource adapter. This property is required.
6. Set the class path. The list of paths or JAR file names that together form the location for the resource adapter classes is set here. This includes any additional libraries needed by the resource adapter. The resource adapter code base is automatically added to the class path, but if anything outside the RAR is needed it can be specified here.
7. Set the native path. The list of paths that form the location for the resource adapter native libraries is set here. The resource adapter code base is automatically added to the class path, but if anything outside the RAR is needed it can be specified here.
8. Set the ThreadPool alias. The name of a thread pool that is configured in the server that is used by the resource adapter Work Manager is specified in this field. If there is no thread pool configured in the server with this name, the default configured thread pool instance, named Default, is used. This property is only necessary if this resource adapter uses Work Manager. This field does not apply for the z/OS® platform.

Resource adapters collection

Use this panel to perform the following actions on stand-alone resource adapters: view the list of installed resource adapters, install additional resource adapters, create additional configurations of already installed resource adapters and delete resource adapter configurations.

A resource adapter can be provided as a stand-alone adapter or as part of an application, in which case the resource adapter is referred to as an embedded adapter. Refer to related task, Installing resource adapters within applications, for more information on embedded resource adapters. A resource adapter is an implementation of the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) specification. Enterprise applications can use a resource adapter to access resources outside of the application server including relational databases like DB2, online transaction processing (OLTP) systems like CICS, and enterprise information system (EIS) like SAP and PeopleSoft. A resource adapter can provide an EIS with the ability to communicate with message-driven beans (MDB) that are configured on the server. Resource adapters are provided by IBM or third party vendors. A resource adapter implementation is provided in a resource adapter archive file; this file has an extension of `.rar`.

To view this administrative console page, click **Resources** > **Resource Adapters** > **Resource adapters**.

To display a list of all of the resource adapters that are defined for a specific scope, select that scope.

To view the stand-alone resource adapters that are provided with the application server, select the **Show built-in resources** checkbox in the **Preferences** section.

To view additional information about, or to change the settings of a specific resource adapter, click the resource adapter name.

To perform an action on a specific resource adapter, select the checkbox beside the resource adapter name and click the appropriate button detailed below.

Install RAR

Install a resource archive (RAR).

You can upload a RAR file from the local file system, or specify a RAR file on a remote file system. The RAR file must be installed at the node level.

New

Create a copy of the selected resource archive which is already installed on the application server.

If you want to create a copy of an installed resource adapter, specify a server for the scope, and click **New**. You cannot create a copy of a resource adapter at the node scope. If you want to install a new resource adapter, click **Install RAR**.

Delete

Delete the selected resource adapter.

Update RAR

Update the selected resource adapter. Update a resource adapter archive (RAR) file when you determine that a resource adapter, or a set of resource adapters, needs to be updated with a different version or implementation.

Different versions or implementations of resource adapters can include different settings, therefore, updating your adapter might be beneficial if you require a specific set of configuration options. You can update the resource adapter for all of the nodes in a cell or all the nodes in a cluster. If some of your nodes are earlier than Version 7.0, the RAR update is not supported until those nodes are migrated to Version 7.0 or later.

Name
Specifies the name of the resource adapter.

Description
Specifies a text description of the resource adapter.

This description is a free-form text string to describe the resource adapter and its purpose.

Scope
Specifies the level at which this resource adapter is visible. For general information, read about administrative console scope settings.

Some considerations that you should keep in mind for this particular panel are:

- Changing the scope enables you to see which resource adapter definitions exist at that level.
- Changing the scope does not have any effect on installation. Installations are always done under a scope of node, no matter what you set the scope to.
- When you create a new resource adapter from this panel, you must change the scope to what you want it to be before you click **New**.

Resource adapter settings
Use this page to specify settings for a resource adapter.

A resource adapter is an implementation of the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) specification that provides access for applications to resources outside of the server, provides access for applications to an enterprise information system (EIS), or provides access for an EIS to applications on the server. Resource adapters provide applications access to resources such as DB2, CICS, SAP and PeopleSoft. Resource adapters can provide an EIS with the ability to communicate with message driven beans that are configured on the server. Some resource adapters are provided by IBM; however, third party vendors can provide their own resource adapters. A resource adapter implementation is provided in a resource adapter archive file (RAR); this file has an extension of `.rar`. A resource adapter can be provided as a stand alone adapter or as part of an application, in which case the resource adapter is referred to as an embedded adapter.

The JCA Version 1.6 specification adds support for Java annotations in RAR modules. For more information on annotation support see the topic, JCA 1.6 support for annotations in RAR modules.

To view this administrative console page, click one of the following paths:

- **Resources** > **Resource Adapters** > **Resource adapters** > **New**.
- **Resources** > **Resource Adapters** > **Resource adapters** > ***resource_adapter***.
- **Applications** > **WebSphere enterprise applications** > ***enterprise_application*** > **Manage Modules** > ***connector_module*** > **Resource Adapter**.
- Install a new resource adapter archive:
  1. Click **Resources** > **Resource Adapters** > **Resource adapters** > **Install RAR**.
  2. Specify a full path for the local file system or remote file system, and click **Next**.

Scope:

Specifies the highest topological level at which application servers can use this adapter.

The Scope field is a read-only string field that specifies where the particular definition for a resource adapter is located. The Scope field is set when the resource adapter is installed, which can only be at the node level, or when a new resource adapter definition is added.

Name:

Specifies the name of the resource adapter definition.

This property is a required string containing no spaces that is a meaningful text identifier for the resource adapter.

Description:

Specifies a text description of the resource adapter.

This description is a free-form text string to describe the resource adapter and its purpose.

Archive path:

Specifies the path to the installed resource archive file that contains the module for this resource adapter.

You can only select RAR files that are installed on the nodes within the selected scope, preventing you from configuring a selection that might fail for some of your nodes.

Note: For resources at the cell scope, the RAR files that are available are those that are installed on each individual node in the entire cell. For resources at a cluster scope, the RAR files that are available are those that are installed on each individual node in that particular cluster.

This property is required.

Data type String

Class path:

Specifies a list of paths or Java archive file (JAR) names that together form the location for the resource adapter classes.

Class path entries are separated by using the ENTER key and must not contain path separator characters like ';' or ':'. Class paths can contain variable (symbolic) names that can be substituted using a variable map. Check your driver installation notes for specific JAR file names that are required.

Native library path:

Specifies an optional path to any native libraries, which are .dll or .so files.

Native path entries are separated by using the ENTER key and must not contain path separator characters like ';' or ':'. Native paths can contain variable (symbolic) names that can be substituted using a variable map.

Isolate this resource provider:

Specifies that this resource provider will be loaded in its own class loader. This allows different versions of the same resource provider to be loaded in the same Java Virtual Machine. Give each version of the resource provider a unique class path that is appropriate for that version.

Ensure that all copies of a resource adapter have the same value for this option. For example, if you create a resource adapter at the cluster scope, the value of this option will be taken from the resource adapter archive (RAR) that you copy. When you create the copy, you cannot modify the value for any instances of that RAR, which would be the copies at the node or cluster scope in this example. If you need to modify the value, you have to delete the copies of the RAR until there is only one instance of that particular RAR that is left.

Note: You cannot isolate a resource provider if you specify a native library path.

Advanced resource adapter properties

Use this page to specify advanced settings for resource adapters that comply with the Version 1.5 and 1.6 Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) specification.

A resource adapter is an implementation of the JCA specification that provides access for applications to an enterprise information system (EIS), like DB2, CICS, SAP and PeopleSoft, or provides access for an EIS to applications on the server. A resource adapter can also provide an EIS with the ability to communicate with message-driven beans that are configured on the server. Some resource adapters are provided by IBM, but third party vendors can provide their own resource adapters.

A resource adapter implementation is provided in a resource adapter archive file; this file has an extension of .rar. A resource adapter can be provided as a stand-alone adapter or as part of an application, in which case it is referred to as an embedded adapter.

To view this administrative console page, click **Resources** > **Resource Adapters** > **Resource adapters** > ***resource_adapter*** > **Advanced resource adapter properties**.

Restrict the JVM to allow only one instance of this resource adapter:

Prevents more than one instance of a resource adapter JavaBeans with a unique resource adapter implementation class name from existing in the same Java Virtual Machine (JVM). This field is only available on resource archives that allow definitions for activation specifications.

Note: Enabling this setting imposes a restrictive condition on the inbound communications. For example, if two applications embed the same resource adapter, only the first application to start will be able to access resources through its embedded resource adapter. If a stand-alone resource adapter is configured for a single instance, no applications that embed that same resource adapter will be able to access resources.

Data type	Boolean (checkbox)
Default	False (disabled)

Register this resource adapter with the high availability manager:

Specifies that the high availability (HA) manager will manage the lifecycle of a JCA resource adapter in a cluster. This option is only applicable to resource adapters with a version greater than JCA 1.0 and running on the Network Deployment version of WebSphere. Do not select this option without first consulting the product documentation for the resource adapter, because this option requires the resource adapter to support high availability of inbound messaging. This field is only available on resource archives that allow definitions for activation specifications.

Note: Enabling this setting imposes a restrictive condition on the inbound communications.

This setting can be implemented with:

- **Endpoint failover:** allows only one resource adapter in an HA group to receive messages across multiple servers. The result is that only one resource adapter can have endpoints active at one time.
- **Resource adapter instance failover:** allows only one resource adapter in an HA group to be started across multiple servers. Inbound or outbound communication is limited to one resource adapter in the cluster.

Data type	Boolean (checkbox with implementation options)
Default	False (disabled)

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

Default product locations - z/OS

app_server_root
: Refers to the top directory for a WebSphere Application Server node.

  The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are *was.install.root* and *WAS_HOME*.

  The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
: Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

  The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

  The default is /wasv8config/*cell_name*/*node_name*.

plug-ins_root
: Refers to the installation root directory for Web Server Plug-ins.

profile_root
: Refers to the home directory for a particular instantiated WebSphere Application Server profile.

  Corresponding product variables are *server.root* and *user.install.root.*

  In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will always be *app_server_root*/profiles/default because only the profile name "default" is used in WebSphere Application Server for z/OS.

smpe_root
: Refers to the root directory for product code installed with SMP/E or IBM Installation Manager.

  The corresponding product variable is *smpe.install.root*.

  The default is /usr/lpp/zWebSphere/V8R0.

Configuring Oracle Real Application Cluster (RAC) with the application server

Oracle Real Application Cluster (RAC) is a "share-everything" database architecture in which two or more Oracle RAC nodes are clustered together and share the same storage. The RAC nodes are connected together with a high-speed interconnect that enables fast communication between the Oracle nodes. The nodes can exchange various categories of data block ownership information during startup, lock information, exchange transaction information and data, and so on.

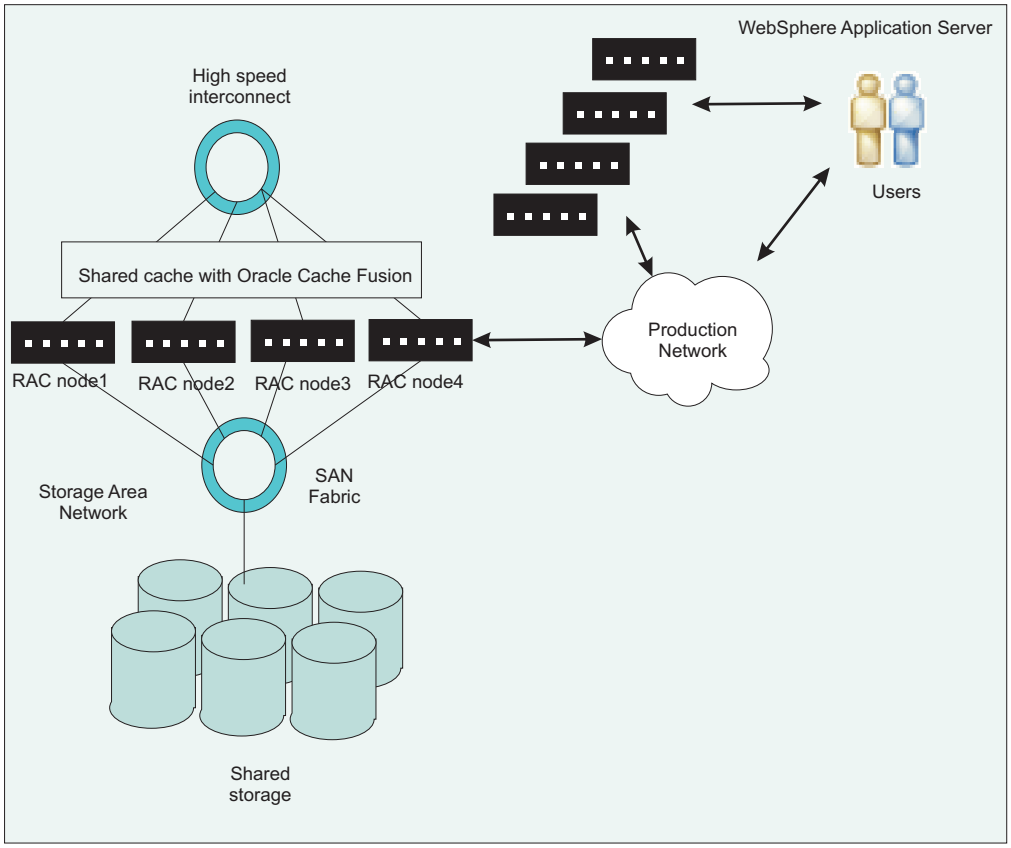
About this task

Using the Oracle JDBC driver, you can configure failover support, load balancing, or both, in an Oracle Real Application Clusters (RAC) environment. Oracle RAC is an option of an Oracle database that brings together two or more computers to form a clustered database that behaves as a single system. In a RAC database, Oracle processes that are running in separate nodes access the same data from a shared disk storage. First introduced in Oracle Version 9i, RAC provides both high availability and flexible scalability.

A typical Oracle RAC cluster consists of the following:

- **Cluster nodes** – 2 to *n* nodes or hosts, running the Oracle database server.
- **Network Interconnect** – a private network used for cluster communications and cache fusion. This is typically used for transferring database blocks between node instances.
- **Shared Storage** – used to hold the database system and data files. The shared storage is accessed by the cluster nodes.
- **Production network** – used by clients and application servers to access the database.

The following figure depicts a typical configuration for Oracle RAC:



Here are two of the many features that Oracle RAC provides:

- *Oracle Notification Service (ONS)* allows for Oracle RAC to communicate the status for the nodes, which are typically UP and DOWN events, to the Oracle JDBC driver and the driver's connection cache. To take advantage of ONS, you must configure the application server to use Oracle's connection caching instead of the application server's connection pooling feature. Read the topic Configuring Oracle connection caching in the application server for more information on this process.
- *Distributed Transaction Processing (DTP)* is a feature that was introduced in Oracle 10gR2. When this feature is enabled, Oracle will ensure that all in-flight prepared transactions that belong to a DTP service for failed RAC instances are pushed to disk. Then, Oracle will restart the DTP service on any of the RAC instances that are still operational.

For more information on Oracle RAC and how it works with the application server, refer to *Building a high availability database environment using WebSphere middleware: Part 3: Handling two-phase commit in WebSphere Application Server using Oracle RAC* on the developerWorks® website.

Procedure

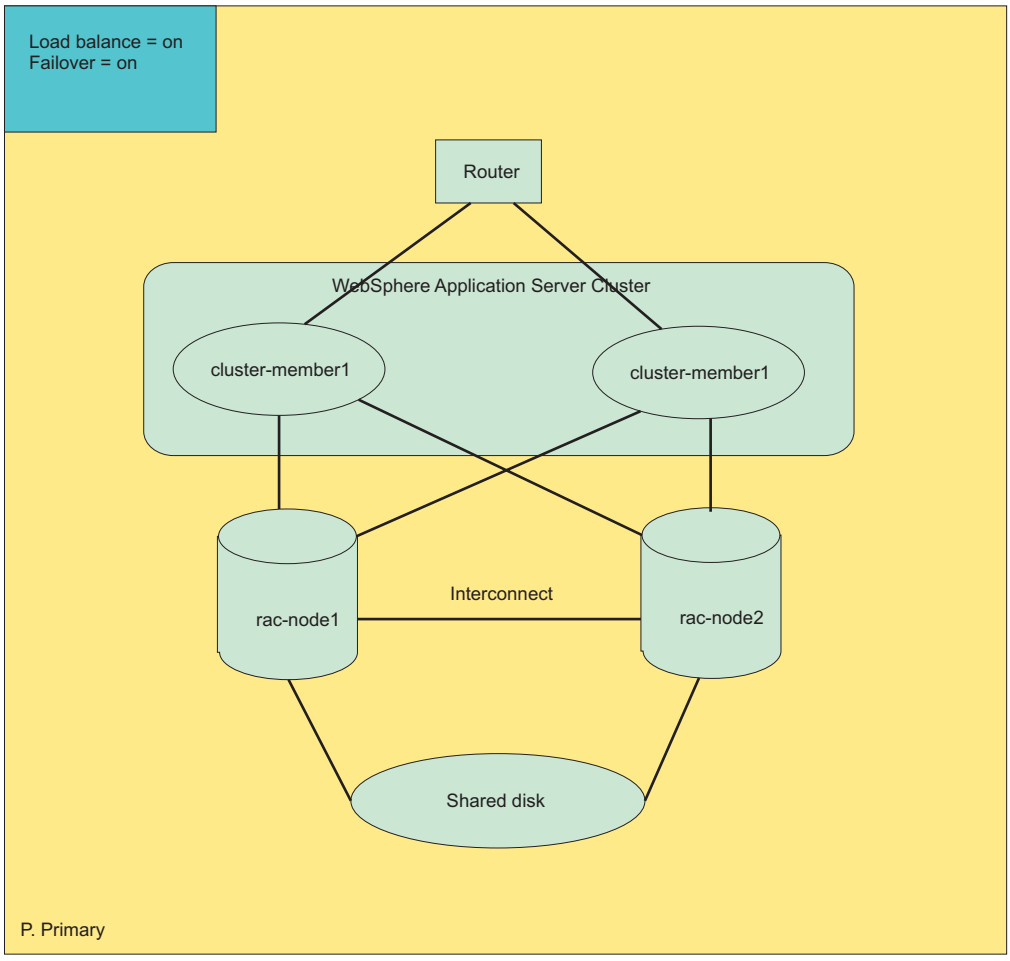
- “Configuring a simple RAC configuration in an application server cluster.”
- “Configuring Oracle connection caching in the application server” on page 10.
- “Configuring two-phase commit distributed transactions with Oracle RAC” on page 12.

Configuring a simple RAC configuration in an application server cluster

Oracle Real Application Cluster (RAC) is a "share-everything" database architecture that can provide high availability and load balancing. A typical configuration for an Oracle RAC contains two or more Oracle RAC nodes that are clustered together and share the same storage.

About this task

This figure depicts a typical RAC physical topology in a cluster environment for the application server, and both the failover and load balancing are enabled:



In the figure above, the application server cluster consists of two members: cluster-member1 and cluster-member2. The Oracle RAC physical configuration contains two nodes: rac-node1 and rac-node2. The RAC nodes can be located in the same physical machine with the cluster members, or they could be placed in entirely different machines. The actual placement does not impact the fundamental qualities of the services provided by RAC. To achieve both high availability and load-balancing, you can specify the Oracle data source URL for both cluster members in the application server with the required properties.

Procedure

1. Navigate to the Oracle data source. Click **Resources** > **JDBC** > **Data sources** > ***oracle_data_source***. If you don't already have an Oracle data source, create a new data source by clicking **New** and completing the wizard. For the URL, substitute the properties in the next step.
2. Set the URL for the Oracle database with the required configuration parameters.

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
                  (ADDRESS=(PROTOCOL=TCP)(HOST=rac-node1)(port=1521))
                  (ADDRESS=(PROTOCOL=TCP)(HOST=rac-node2)(port=1521)))
                  (FAILOVER=on)(LOAD_BALANCE=on)
                  (CONNECT_DATA=(SERVER=DEDICATED)
                  (SERVICE_NAME=<service_name>)))
```

   Note: Be aware of these configuration options:
   - If you are not using Oracle services, then *service_name* will be the database name in the example. If you are using Oracle services, then *service_name* will be the name of the services.
   - The example has FAILOVER and LOAD_BALANCE turned on. To turn one or both of these features off, change `on` to `off` in the above example.
3. Click **Apply** or **OK**.

Configuring Oracle connection caching in the application server

You can elect to configure an Oracle data source to use the Oracle connection caching feature instead of using the application server connection pooling. Connection caching for Oracle databases is similar to connection pooling in the application server.

About this task

Currently, Oracle supports connection caching only with data sources that use the oracle.jdbc.pool.OracleDataSource implementation class, instead of the oracle.jdbc.pool.OracleConnectionPoolDataSource or oracle.jdbc.xa.client.OracleXADataSource classes. By default, the Oracle JDBC providers in the application server are configured to use the oracle.jdbc.pool.OracleConnectionPoolDataSource for non-XA data sources, or oracle.jdbc.xa.client.OracleXADataSource for XA data sources. To enable Oracle connection caching, you must configure and use a new JDBC provider in the application server that implements the oracle.jdbc.pool.OracleDataSource class.

Note: Oracle connection caching does not support XA.

Procedure

1. Create a data source and user-defined JDBC provider.
   a. Click **Resources** > **JDBC** > **Data sources**
   b. Select a server from the **Scope** drop-down list.
   c. Click **New**.
   d. Enter the name and JNDI name for the data source. Click **Next**.
   e. Create a JDBC provider. Select **Create new JDBC provider**, and click **Next**.
   f. Define the required properties for the JDBC provider. Use the following configuration settings:
      - **Database type:** User-defined
      - **Implementation class name:** oracle.jdbc.pool.OracleDataSource

      Click **Next**.
   g. Enter the class path for `ojdbc6.jar`, and click **Next**.
   h. For **Data store helper class name**, enter `com.ibm.websphere.rsadapter.Oracle11gDataStoreHelper`. Click **Next**.

i. Define the security aliases for this data source, and click **Next**.
j. Finish the wizard.
k. Save the configuration changes.

2. Configure the data source that you created.
   a. Click the name of the data source. The configuration panel displays.
   b. Select **Custom properties**, and create or modify the properties for this data source. Enter or update the following custom properties:

Name	Value
disableWASConnectionPooling	true **gotcha:** You must also set the maximumPoolSize attribute to `0` on WebSphere Application Server connection pool settings to allow Oracle to control the pool boundaries.
connectionCachingEnabled	true
connectionCacheName	*your_cache_name*
removeExistingOracleConnectionPoolIfExists	true **Note:** The removeExistingOracleConnectionPoolIfExists property must be set to `true` so the application server removes any existing Oracle connection pools with an identical name. Otherwise, the Oracle data source fails the getConnection method if the pool name that is created has a name that is identical to an existing pool. For example, if you run a test connection, the test connection process creates an Oracle connection pool that prevents the application server from working properly at run time.
URL	*Oracle_URL*

Note: The order in which the custom properties are set is important. The setting order can be an issue because the application server passes the properties as a collection and the order is not guaranteed. If you encounter this issue, contact Oracle and reference Oracle bug #6638862.

3. Click **Apply** or **OK**.
4. Save the changes to the application server configuration.
5. Restart the application server.

Results

Oracle does not display a message if the pool creation fails, and a normal connection is returned instead. You can confirm that the Oracle connection pool is created by using the administrative console test connection function for the data source. First, turn on trace with the trace string, "RRA=all", for the server that runs your application. Then, issue a test connection. Issue a second test connection. Both test connections should work. Examine the trace log.

If the Oracle connection pool was created successfully, the trace shows that the second test connection detected that the Oracle connection cache exists because of the first test connection, and was successful in removing it so that it can be created again by the second test.

Configuring two-phase commit distributed transactions with Oracle RAC

Real Application Cluster (RAC) configurations for Oracle 10g have an inherent issue with the transaction manager when Oracle attempts to recover two-phase commit distributed transactions that span over multiple Oracle RAC nodes. A problem can occur when one node fails, and Oracle opens up the other surviving nodes for business before the Oracle RAC completes the necessary recovery action for the node that has failed. The application server's ability to maintain transaction affinity provides you the ability to circumvent this issue.

About this task

Errors can occur when the recovery process attempts to commit or rollback a transaction branch through a RAC node that was previously active but later failed. The transaction manager would receive the following exception:

```
ORA- 24756: transaction does not exist
```

If this error is encountered, the Oracle database administrator might need to manually resolve the in-doubt transaction by forcing a rollback or commit process. If you do not desire a manual intervention, however, you might want to configure an automatic and transparent strategy for transaction recovery.

If the in-doubt transaction is not resolved, any subsequent transactions will receive the following exception:

```
ORA-01591 lock held by in-doubt distributed transaction
```

The result is that portions of the database will not be usable.

The key to a transparent recovery strategy is to eliminate the possibility of a global transaction spanning more than one transaction branch over multiple RAC nodes. A transaction branch corresponds to a database connection that is enlisted in a global transaction. If all connections in a global two-phase commit transaction originate from the same node, transaction recovery problems should not arise. Configure an Oracle RAC with the application server to prevent errors with two-phase transactions.

The application server maintains transaction affinity for incoming connections, and you can take advantage of this feature to configure automatic recovery for Oracle RAC with two-phase commit transactions. If you implement this configuration, all connections from a given application server will be received from the same Oracle node, and the connections will finish on that same node. This configuration will avoid situations in which transactions span multiple nodes, and you should not experience a recovery problem if one or more Oracle nodes go down.

Procedure

- You can elect to manually resolve the in-doubt transaction.
  1. Get the orphaned transaction ID. Issue the following command:

     ```
     sql > select state, local_tran_ID, Global_tran_Id from dba_2pc_pending where state = "prepared"
     ```
  2. Roll back all of the transaction IDs that are in the prepared phase.

     ```
     sql > rollback force '';
     ```
- Configure an automatic strategy for transaction recovery.
  1. Create an Oracle service that has only one primary node. Creating the service with one primary node will ensure that load balancing is disabled. You can also specify one or more alternate nodes with the -a parameter. Run this command to create the service:

     ```
     srvctl add service -d <database_name> -s <service_name> -r <primary nodes> -a <alternate_nodes>
     ```
  2. Enable Distributed Transaction Processing (DTP) on the Oracle service. DTP was first introduced in Oracle 10gR2. Each DTP service is a singleton service that is available on only one Oracle RAC instance. Run this command:

     ```
     execute dbms_service.modify_service (service_name => '<service_name>' , dtp => true);
     ```

3. Configure each cluster member in the application server to use the Oracle DTP service.

Results

If you configured an automatic recovery strategy, the DTP service will start automatically on the preferred instance. However, if the database is restarted, the DTP service will not start automatically. You can start the DTP service using this command:

```
srvctl start service -d  -s
```

If a RAC node stops working, Oracle will not failover the DTP service until the Oracle RAC cleanup and recovery is complete. Even if the Oracle nodes come back up, the Oracle DTP service will not return to the freshly restarted RAC node. Instead, you will have to manually move the service to the restarted RAC node.

When you configure DTP on the Oracle service, you have transferred load balancing from the Oracle JDBC provider to the application server. The workload will be distributed by the application server instead of Oracle, which is why you created services that do not implement load balancing and only use one primary node. This configuration prevents situations in which transaction processes span multiple RAC nodes and alleviates recovery problems that can arise when one or more RAC nodes fail.

Configuring client reroute for applications that use DB2 databases

The client reroute feature enables you to configure your client applications for a DB2 universal database to recover from a communication loss, and the applications can continue to work with minimal interruption. Rerouting is central to the support of continuous operations, but rerouting is only possible when there is an alternate location that is identified to the client connection.

Before you begin

This task assumes the following:

- You have a DB2 data source defined in the application server. See the topic, Configuring a data source using the administrative console, for information about creating a data source.
- The DB2 data source to which your application connects is running one of the following:
  - DB2 for z/OS Version 9.1 or later
  - DB2 Database for Linux, UNIX, and Windows Version 9.5 or later
- You have implemented the DB2 database with a redundant setup or the ability to fail the DB2 server to a standby node.
- You are connecting to the data source with a Type-4 connection.

About this task

Client reroute for DB2 allows you to provide an alternate server location, in case the connection to the database server fails. If you decide to use client reroute with the persistence option, the alternate server information persists across Java Virtual Machines (JVMs). In the event of an application server crash, the alternate server information is not lost when the application server is restored and attempts to connect to the database.

Without any configuration on the client side, a JDBC driver for DB2 supports the client reroute capability, if it is enabled, when the driver makes an initial connection to the DB2 server. When the JDBC driver connects to a DB2 server that has an alternate server configured, the primary server sends information about the alternate server to the JDBC driver. If the connection to the primary server fails, the JDBC driver is able to reroute connections to the alternate server. If the client process crashes, however, the alternate

server information is lost, and the client needs to connect to the primary server again. If the client cannot make an initial connection to the primary server, the client has no knowledge of the alternate server and cannot reroute.

To overcome this problem, you can configure a DB2 data source in the application server with the **Alternate server name** and **Alternate port number** fields, or with the clientRerouteAlternateServerName and clientRerouteAlternatePortNumber data source custom properties, to support client reroute even on the initial connection attempt. If the JDBC driver is not able to connect to the primary DB2 server, the information that is necessary for a client reroute is already present, and the JDBC driver can reroute the connection to an alternate server.

Attention: The data source custom property, enableClientAffinitiesList, changes the semantics of the clientRerouteAlternateServerName and clientRerouteAlternatePortNumber properties.
To learn more about these properties, see the DB2 information center topic, Common IBM Data Server Driver for JDBC and SQLJ properties for all supported database products. To learn more about client affinity, see the topic, .Configuring client affinity for applications that use DB2 databases.

Additionally, if you have configured a DB2 data source as a Type 4 JDBC driver, you can use the **Client reroute server list JNDI name** field, or the clientRerouteServerListJNDIName data source custom property, to enable persistence of the client reroute state. Typically, when a connection is rerouted and the JDBC driver has connected to the alternate DB2 server, the alternate server sends information about its own alternate server to the JDBC driver. The JDBC driver will then have the information that is required to reroute the connection again if the alternate DB2 server is not available. Effectively, the server that was originally the alternate server is now the primary server, and a new alternate server has been established. If you enable persistence for client reroute, this new state can be remembered. If the application server crashes and is restarted, the JDBC driver can connect to the DB2 server that was considered the primary server at the time of the crash. Without the persistence feature, the JDBC driver would have to start from the original server configuration and attempt to connect to the server that was originally considered the primary server.

You can use the automatic client rerouting feature within the following DB2 configurable environments:

- Enterprise Server Edition (ESE) with the data partitioning feature (DPF)
- Data Propagator (DPROPR)-style replication
- High availability cluster multiprocessor (HACMP™)
- High availability disaster recovery (HADR).

Procedure

1. In the administrative console, click **Resources** > **JDBC** > **Data sources** > ***data_source***.
2. Click **WebSphere Application Server data source properties**.
3. In the **DB2 automatic client reroute options** section, fill in the fields to enable client rerouting. Complete the following fields:

   Alternate server names
   Specifies the list of alternate server name or names for the DB2 server. If more than one alternate server name is specified, the names must be separated by commas. For example:

   ```
   host1,host2
   ```

   Alternate port numbers
   Specifies the list of alternate server port or ports for the DB2 server. If more than one alternate server port is specified, the ports must be separated by commas. For example:

   ```
   5000,50001
   ```

   Note: Ensure that an equal number of entries must be specified for both alternate ports and hosts. Otherwise, a warning is displayed and client reroute is not enabled.
4. Optional: Enable client reroute with the persistence option.

a. Complete the field for **Client reroute server list JNDI name**. The field specifies the JNDI name that is used to bind the DB2 client reroute server list into the JNDI name space. The DB2 database server uses this name to look up the alternate server name list when the alternate server information is not already in memory.

   Note: Be aware of the following:
   - This option is not supported for Type 2 data sources. If you use a DB2 data source that is configured as a Type 2 JDBC driver, the JDBC driver uses a catalog to persist the client reroute information. If this property is configured with a Type 2 driver, the application server will issue a warning.
   - Use different JNDI names among different data sources. Otherwise, when you delete a data source, and the JNDI entry is removed from the name space, the other data sources that share the JNDI entry will be affected.

5. Configure the retry count and interval for the client reroute function. Complete these two fields:

   Retry interval for client reroute
   Specifies the amount of time, in seconds, between retries for automatic client reroute.

   Maximum retries for client reroute
   Specifies the maximum number of connection retries that are attempted by the automatic client reroute function if the primary connection to the server fails. The property is only used when **Retry interval for client reroute** is set.

   Attention: If you do not specify a value for these properties, DB2 failover processing (client rerouting) does not occur.

6. Click **OK** and save the changes.
7. Restart the application server.

What to do next

If you later want to remove the client reroute information that is bound in JNDI, you can do so by deleting the data source. You can also use the unbind feature with the test connection service to delete the JNDI binding for the client reroute function from the application server's JNDI name space without deleting the data source.

To delete the JNDI binding for client reroute:

1. Select **Unbind client reroute list from JNDI**.
2. Click **OK**.
3. Save the configuration.
4. Click **Test connection** for the data source.
5. Deselect **Unbind client reroute list from JNDI**.
6. Click **OK**.
7. Save the configuration.

Configuring connection validation timeout

You can configure a timeout for connection validation by the Java Database Connectivity (JDBC) driver through a data source custom property in the data source configuration panels.

About this task

You can choose between validating connections with the JDBC driver or by having the application server run a SQL query. Select one or both of the following connection pretest attributes:

- Validate new connections

- Validate existing pooled connections

By default, connection validation is disabled. When you save the configuration for the data source, the administrative console supplies only the option that is selected. The administrative console will select validation by timeout or validation by a query, but if validation is not enabled then the application server will select neither option.

Procedure

1. Open the administrative console.
2. Go to the **WebSphere Application Server Data Source properties** panel for the data source.
   a. Select **Resources** > **JDBC** > **Data Sources** > ***data_source***
   b. Select **WebSphere Application Server Data Source properties**.
3. Go to the **Connection Validation Properties** section.
4. Select the type of connections that the application server will validate.
   - Select **Validate new connections**. This option specifies that the connection manager tests newly created connections to the database.
   - Select **Validate existing pooled connections**. This options specifies that the connection manager tests the validity of pooled connections before returning them to applications.
   - You can also select both options

     Note: You must make a selection here. If you do not select one or both of these options, you will not be able to select **Validation by JDBC Driver**. The **Validation by JDBC Driver** timeout feature is only available for JDBC providers that comply with the JDBC 4.0 specification.

     For an Oracle datasource, **Validation by JDBC Driver** appears on the administrative console only after the validateNewConnectionTimeout property is added to the custom properties of WebSphere Application Server datasource properties. The validateNewConnectionTimeout property is used for JDBC 4.0 driver validation and can be specified using administrative console.

   .
5. Click **Validation by JDBC Driver**. The application server issues a warning if **Validation by JDBC driver** is configured and the JDBC driver does not implement JDBC 4.0, or if the Connection.isValid method raises an error.

   Note: Connection validation by SQL query is deprecated. Use validation by JDBC Driver instead.
6. Enter the timeout value in the input box. The timeout value is in seconds.

   Note: If retries are configured, meaning the retry interval is not set to 0, for **Validate new connections** or **Validate existing pooled connections**, then the full value of the timeout applies to each retry. For each retry, the application server waits for the retry interval. Then the JDBC driver uses the full value of the timeout to validate the connection
7. Save the data source configuration.

What to do next

If you are modifying an existing data source, restart your server for this change to go into effect. If this is a new data source, restarting the server is not necessary.

Chapter 2. Establishing high availability for Service integration

This page provides a starting point for finding information about service integration.

Service integration provides asynchronous messaging services. In asynchronous messaging, producing applications do not send messages directly to consuming applications. Instead, they send messages to destinations. Consuming applications receive messages from these destinations. A producing application can send a message and then continue processing without waiting until a consuming application receives the message. If necessary, the destination stores the message until the consuming application is ready to receive it.

High availability and workload sharing for service integration technologies

These topics provide information about high availability and workload sharing for service integration technologies.

About this task

- High availability and workload sharing
- “Configuring high availability and workload sharing of service integration”
- “Administering high availability for service integration” on page 25
- “Managing high availability when messaging engines fail to start” on page 26

Configuring high availability and workload sharing of service integration

You can configure high availability and workload sharing of service integration without using messaging engine policy assistance.

Before you begin

Ensure that you want to use the following procedure. As an alternative, you can configure high availability and workload sharing of service integration by using messaging engine policy assistance when you add a server cluster to a bus. You create messaging engines and their associated policies as part of the procedure, by using the appropriate predefined messaging engine policy type. Alternatively, you can use a custom policy type and configure the messaging engine policy as you require, and the relevant core group policies and match criteria are created automatically.

About this task

When you set up a service integration environment, you create bus members, either servers or clusters, that run messaging engines. For high availability, where the messaging engine can fail over, or workload sharing, where multiple messaging engines share the load on a destination, you need to create a cluster bus member and configure high availability and workload sharing characteristics of the messaging engines.

If you do not require high availability or workload sharing, you can use a simple configuration and create a server bus member. You do not need the steps described in this topic.

The high availability and workload sharing characteristics of the messaging engines in the cluster are set by core group policies.

To see the policies that are configured in your system, you can use the administrative console to open the Policies page. In the navigation pane, click **Servers -> Core groups -> Core group settings -> *core_group_name* -> [Additional Properties] Policies**.

One of the available policies is the default service integration policy, "Default SIBus Policy", which is the policy that a messaging engine uses unless you configure the system so that the engine uses another policy. The default policy is sufficient for many purposes and you might not need to alter the policy configuration. It is not advisable to alter the default service integration policy, because those changes will affect all messaging engines that the policy manages. Therefore, it is better to create and configure one or more new specific policies.

Procedure

1. Optional: Create a cluster, if it is not created already. See Creating clusters.
2. Add the cluster to the service integration bus. See Adding a cluster to a bus without using messaging engine policy assistance.

   A single messaging engine that uses the default service integration policy is created automatically.

   For high availability without workload sharing, you can use this configuration and do not need to change it further. If you want to configure the messaging engine behavior further, for example to specify preferred servers for the messaging engine, or enable the messaging engine to fail back, complete step 3.
3. Optional: For high availability when you want to configure the messaging engine behavior, create and configure a policy for the messaging engine. Create a policy with the type “One of N” . See “Creating a policy for messaging engines” on page 19 and “Configuring a core group policy for messaging engines” on page 20.
4. Optional: For workload sharing without high availability, use the following steps:
   a. Add as many messaging engines as you require to the cluster. Typically, a workload sharing configuration has one messaging engine for each server in the cluster. See Adding a messaging engine to a cluster.
   b. Create and configure a policy for each messaging engine in the cluster. Create policies with the type Static. See “Creating a policy for messaging engines” on page 19 and “Configuring a core group policy for messaging engines” on page 20.
5. Optional: For workload sharing with high availability, use the following steps:
   a. Add as many messaging engines as you require to the cluster. Typically, a workload sharing configuration has one messaging engine for each server in the cluster. See Adding a messaging engine to a cluster.
   b. Create and configure a policy for each messaging engine in the cluster. Create policies with the type “One of N”. See “Creating a policy for messaging engines” on page 19 and “Configuring a core group policy for messaging engines” on page 20.
6. Optional: To use an external high availability (HA) framework to manage high availability or workload sharing behavior, use the following steps:
   a. If you require workload sharing, add as many messaging engines as you require to the cluster. Typically, a workload sharing configuration has one messaging engine for each server in the cluster. See Adding a messaging engine to a cluster.
   b. Create and configure one policy for the messaging engines in the cluster. Create a policy with the type “No operation”. See “Creating a policy for messaging engines” on page 19 and “Configuring a core group policy for messaging engines” on page 20.

What to do next

If you created a high availability configuration for service integration, you might also want to configure high availability for the transaction service.

If you created a workload sharing configuration, you might want to deploy a queue destination to the cluster, so that the queue is partitioned across the set of messaging engines.

Creating a policy for messaging engines

You create one or more core group policies for service integration to control the behavior of the messaging engine, or engines, in a server cluster. The policies support behavior such as high availability, workload sharing or scalability in a server cluster.

Before you begin

Ensure that you want to use the following procedure. As an alternative, you can create a policy by using messaging engine policy assistance when you add a server cluster to a bus. You create messaging engines and their associated policies as part of the procedure, and use predefined messaging engine policy types that support frequently-used cluster configurations. Alternatively, you can use a custom policy type and configure the messaging engine policy as you require, and the relevant core group policies and their match criteria are created automatically.

Continue with the following procedure to create a core group policy for messaging engines if you are familiar with it. Otherwise, it is easier to create a policy by using messaging engine policy assistance when you add a server cluster to a bus.

Decide what type of core group policy you need to create for the configuration you require. For service integration, the following types of core group policy apply:

- Static. Use this type of policy for a workload sharing or scalability configuration without high availability. Create one policy for each messaging engine in the cluster.
- One of N. Use this type of policy for a high availability configuration, or a workload sharing configuration with high availability. You create one policy for each messaging engine in the cluster.
- No operation. Use this type of policy when you use an external high availability framework to manage the messaging engines in the cluster. You create one policy for all the messaging engines in the cluster. The configuration might be high availability, or workload sharing with high availability.

For further information, see Policies for service integration.

About this task

A policy is a component of a core group. A core group can have a number of different policies; each policy applies to a particular high availability group and determines the high availability behavior of resources in that group. For service integration, the resources that you want to control are the messaging engines. Typically, you create one policy for each messaging engine in the cluster, unless you want the messaging engines to be managed by external high availability framework.

To create a policy for a messaging engine, use the administrative console to complete the following steps.

Procedure

1. In the navigation pane, click **Servers -> Core groups -> Core group settings -> *core_group_name* -> [Additional Properties] Policies**. A list of currently configured core group policies is displayed.
2. Click **New**.
3. Select one of the following options from the **Policies** list. Only the following policy types are applicable to service integration:

   Static A messaging engine cannot fail over in a WebSphere Application Server cluster.

   One of N
   A messaging engine can fail over in a WebSphere Application Server cluster.

No operation

A messaging engine is managed by an external high availability framework such as IBM HACMP. This option is for use with an external high availability cluster.

Do not select any other policy type, because they because they are not supported for the service integration bus component.

4. Click **Next**. The policies configuration page is displayed.
5. Enter a **Name** that is unique in the scope of the core group.
6. Click **Apply** or **OK**.
7. Configure the policy. See “Configuring a core group policy for messaging engines.”
8. Save your changes to the master configuration.

Configuring a core group policy for messaging engines

You can configure a core group policy for service integration to associate the policy with specific messaging engines and to specify messaging engine behavior, such as which server a messaging engine runs on, whether a messaging engine can fail over or fail back, and the frequency of messaging engine monitoring.

Before you begin

The policy that you want to associate with the messaging engine must exist. It is possible to configure the default service integration policy, "Default SIBus Policy", but it is not advisable, because those changes will affect all messaging engines that the policy manages. Therefore, if you want to configure a policy, it is advisable to create a new one, as described in “Creating a policy for messaging engines” on page 19.

About this task

Only the following types of core group policy apply to service integration and messaging engines:

- Static
- One of N
- No operation

You cannot use the “All active” or “M of N” policy types, because they are not supported for the service integration bus component.

The default service integration policy is a “One of N” policy with a single match criterion that matches any service integration messaging engine. The default policy has a monitoring interval of 120 seconds, no preferences for particular servers and no automatic fail back. If you want to configure messaging engine behavior, it is advisable to create additional, more specialized policies, and retain the default service integration policy to apply to messaging engines that do not match any other policy.

To configure a core group policy for a messaging engine, or messaging engines, use the administrative console to complete the following steps:

Procedure

1. Select the policy by clicking **Servers -> Core groups -> Core group settings -> *core_group_name* -> [Additional Properties] Policies -> *policy_name***.
2. Associate the policy with the messaging engine, or messaging engines, that you require. Typically, for a policy of type Static or “One of N”, you associate the policy with a single messaging engine. Typically, for a policy of type “No operation”, you associate the policy with all the messaging engines (one or more) in a cluster. See “Using match criteria to associate a policy with a messaging engine” on page 21.
3. Configure the policy further, by using the appropriate procedure for the policy type that you are configuring:

- Configure a Static policy for service integration.
- Configure a “One of N” policy for service integration.
- Configure a “No operation” policy for service integration.

Using match criteria to associate a policy with a messaging engine:

Use this task to configure match criteria to associate a core group policy with a messaging engine.

Before you begin

To complete this task, you must have created a policy to associate with the messaging engine.

About this task

Each messaging engine is managed by an HAGroup to which a policy is assigned at run time. The policy assigned to an HAGroup is chosen by comparing the match criteria of the set of configured policies with the properties of the HAGroup. The policy with the strongest match is assigned to the HAGroup. The following table lists the names and values of the HAGroup properties for a messaging engine, and the set of matching messaging engines if a property is used in the policy match criteria:

Name	Value	The messaging engine or engines that the policy matches
type	WSAF_SIB	Any messaging engine
WSAF_SIB_MESSAGING_ENGINE	The name of the messaging engine. This is in the form *node.server-bus* for a messaging engine in a server, or *cluster.number-bus* for a messaging engine in a cluster, where *number* relates to the order that messaging engines were added to the bus (the first messaging engine that is created when you add the cluster to a bus has the number 000).	A particular messaging engine
WSAF_SIB_BUS	The name of the bus	All messaging engines in a particular bus
IBM_hc	The name of the cluster	All messaging engines in a particular cluster

For more information about match criteria for messaging engines, see Match criteria for service integration.

Note: If you use messaging engine policy assistance to configure the messaging engine behavior for messaging engines in a cluster, suitable match criteria are created automatically and you do not have to specify any.

Procedure

1. Open the match criteria page for your policy by clicking **Servers -> Core groups -> Core group settings -> *core_group_name* -> [Additional Properties] Policies -> *policy_name* -> [Additional Properties] Match criteria**.
2. Click **New** to create a new match criterion.
3. Enter a suitable **Name** and **Value** to specify a messaging engine, or group of messaging engines, that will match this policy. Use the information in the previous table to find the required name and value for the selection you want.

   For more description of the fields on this page, see Match criteria settings. For more information about finding the correct names for match criteria, see Core group settings.
4. Click **OK**.
5. Repeat the previous three steps for each match criteria that you want to add to the policy. You add match criteria to make the match stronger, and to progressively restrict the set of HAGroups that the

policy can match. You must specify at least two match criteria to ensure that the policy creates a stronger match than the match that the “Default SIBus Policy” creates. For example, to associate a policy with all the messaging engines in a cluster, you might specify the following match criteria:

Name	Value
type	WSAF_SIB
WSAF_SIB_BUS	*bus_name*
IBM_hc	*cluster_name*

6. Save your changes to the master configuration.

Configuring a Static policy for service integration:

After you create a new Static core group policy for a messaging engine, you configure the policy to specify which server the messaging engine runs on. Optionally, you can configure the frequency of messaging engine monitoring.

Before you begin

A core group policy with the policy type of Static must exist and you must have first completed the steps in “Configuring a core group policy for messaging engines” on page 20.

About this task

You can use a Static policy to run a messaging engine in either a server or a cluster bus member. A Static policy restricts the messaging engine to a particular server, even in a server cluster. This is useful for a workload sharing configuration, where you want to spread messaging load across multiple servers and failover is not required. You can have multiple messaging engines running in a cluster, with each one restricted to a specific server. Although the Static policy can accept multiple servers, do not configure more than one static group server for use with a messaging engine.

Procedure

1. Specify which server the messaging engine will run on:
   a. Click **Static group servers**.
   b. Select the server you require from the Core group servers list, then click **Add**.
   c. Ensure that there is only one server in the Static group servers list, then click **OK**.
2. Optional: If required, enter a value in the **Is alive timer** field. This value specifies the interval of time, in seconds, at which the high availability manager (HAManager) checks that a messaging engine is running properly. When this value is 0 (zero), the default value of 120 seconds is used.
3. Click **OK**.
4. Save your changes to the master configuration.

Configuring a “One of N” policy for service integration:

After you create a new “One of N” core group policy for a messaging engine, you configure the policy to specify the messaging engine behavior, such as which server the messaging engine runs on, and whether the messaging engine can fail over or fail back. You can also configure the frequency of messaging engine monitoring.

Before you begin

A core group policy with the policy type of “One of N” must exist and you must have first completed the steps in “Configuring a core group policy for messaging engines” on page 20.

About this task

You can use a “One of N” policy to run a messaging engine in a cluster to enable failover. One server in the cluster server runs the messaging engine and other servers in the cluster act as standby servers, ready to run the messaging engine if it cannot run in its current server.

You can use a “One of N” policy to run a messaging engine in a server bus member, but this is equivalent to a cluster with only one server, that is, the value of N is 1, so the messaging engine cannot fail over.

You can configure a “One of N” policy for a messaging engine in a cluster to provide high availability, or workload sharing with high availability, depending on how you set the configuration options. See Policies for service integration.

Procedure

1. Open the Policies page for the policy you are configuring. Click **Servers -> Core groups -> Core group settings -> *core_group_name* -> [Additional Properties] Policies** > ***policy_name***.
2. Optional: If required, define which servers the messaging engine prefers to run on in a preferred servers list:
   a. Under Additional Properties, click **Preferred servers**.
   b. Select the servers you require from the Core group servers list, then click **Add** to add them to the Preferred servers list. Ensure that the servers you select are in the server cluster where the messaging engine runs.
   c. Use **Move up** and **Move down** to adjust the order of the list as required. The earlier a server is in the preferred servers list, the stronger the preference for that server.
   d. Click **OK**.

   The messaging engine runs in the first available server in the preferred servers list and fails over to the next available server in the preferred servers list. If no preferred server is available, the messaging engine can fail over to any other server in the cluster.

   You might use the preferred servers list if one server has more resources available to it or typically performs less work than the others. You might use the preferred servers list to help spread workload across the cluster by configuring multiple policies and specifying a different preferred server for each messaging engine. If you do not define a preferred server list, the messaging engine runs on the first available server in the cluster.
3. Optional: If you defined a preferred servers list, if required, restrict the messaging engine to run only on preferred servers, by selecting the **Preferred servers only** check box. The messaging engine runs in the first available server in the preferred servers list and fails over to the next available server in the preferred servers list. The messaging engine cannot run on a server that is not in the preferred servers list. If no preferred server is available, the messaging engine cannot fail over.

   You can use this option together with a single server in the preferred servers list to restrict a messaging engine to a specific server in a workload sharing configuration. You can use this option together with a preferred servers list to create a configuration that provides workload sharing and aspects of high availability, for example, one primary server and one failover server for each messaging engine. If you require high availability, use this option with care, because you can reduce or remove the high availability of the messaging engine.
4. Optional: If you defined a preferred servers list, if required, specify that the messaging engine automatically fails back to a more preferred server by selecting the **Fail back** check box. If a messaging engine is running on a server that is low in the preferred servers list, or in the cluster but not in the preferred servers list (for example, the messaging engine has failed over), the messaging engine automatically fails back to a more preferred server when one becomes available.
5. Ensure that the messaging engine can always reach its data store or file store. If the messaging engine can fail over, that is, for any configuration with high availability characteristics, the data store or file store must be accessible from any server in the cluster on which the messaging engine might run.

The set of possible servers depends on whether you defined a preferred servers list and whether you selected the **Preferred servers only** option. For example, a configuration might have a cluster of three servers, server1, server2, and server3, with a single messaging engine that uses a policy configured so that the messaging engine can fail over to any of the servers in the cluster. The message store for the messaging engine must be accessible from all three servers. However, if the configured policy specified a preferred server list of server1 and server2, and the **Preferred servers only** option is selected, only server1 and server2 would need access to the data store or file store for that messaging engine.

6. Optional: If required, enter a value in the **Is alive timer** field. This value specifies the interval of time, in seconds, at which the high availability manager (HAManager) checks that a messaging engine is running properly. When this value is 0 (zero), the default value of 120 seconds is used.
7. Click **OK**.
8. Save your changes to the master configuration.

Configuring a "No operation" policy for service integration:

After you create a new "No operation" core group policy for a messaging engine, you can continue the configuration for a messaging engine to be managed by an external high availability (HA) framework. You can also configure the frequency of messaging engine monitoring.

Before you begin

A core group policy with the policy type of "No operation" must exist and you must have first completed the steps in "Configuring a core group policy for messaging engines" on page 20.

About this task

A "No operation" policy allows an external HA cluster to control when and where a messaging engine runs.

Procedure

1. Associate the messaging engine with an externally managed resource group by creating an HA cluster resource for it. Refer to the documentation for your external HA product.
2. Write scripts for the external HA framework to enable it to use the HAManager to start or stop the messaging engine. The scripts invoke operations on the HAManager MBean on the server that is taking ownership of the resource that represents the messaging engine.
3. Ensure that the messaging engine can always reach its data store. If the messaging engine is configured to fail over, its data store must be accessible from any server in the cluster on which it might run. The set of possible servers depends on how you have configured the external HA resource group. A typical configuration is to include the data store as an additional resource in the resource group managed by the external HA cluster. The HA cluster will then ensure that the data store and the messaging engine failover together and remain collocated after the failover. Whether this is the case, or whether a network server is used to make the data store available, the data store must be accessible from any server that might run the messaging engine.
4. If the messaging engine must always be accessible through the same IP address, for example because it is the receiving end of a WebSphere MQ link, you must arrange for that IP address to remain collocated with the messaging engine. You can do this by creating an IP address resource in the same external HA resource group as the resource that represents the messaging engine.
5. If required, enter a value in the **Is alive timer** field. This value specifies the interval of time, in seconds, at which the high availability manager (HAManager) checks that a messaging engine is running properly. When this value is 0 (zero), the default value of 120 seconds is used.

   The "No operation" policy is intended primarily for when you use an external high availability framework such as IBM HACMP. In this situation, you might create a monitoring script that the external framework calls periodically. If you set the value of **Is alive timer** greater than or equal to 0, the HAManager performs health monitoring and the external monitoring script can retrieve the state from the

HAManager MBean. Alternatively, if you set the value of **Is alive timer** to -1, monitoring by the HAManager is disabled, and the external monitoring script can retrieve the state from the messaging engine MBean.

6. Click **OK**.
7. Save your changes to the master configuration.

Configuring messaging engine failover for mixed version clusters

A messaging engine that is hosted on a WebSphere Application Server Version 7.0 or later server cannot fail over to a messaging engine that is hosted on a WebSphere Application Server Version 6 server. If you have a cluster bus member that consists of a mixture of Version 6 and Version 7.0 or later servers, you must make sure the high availability policy is configured to prevent this type of failover.

About this task

To prevent failover of a Version 7.0 or later messaging engine to a Version 6 server, configure the high availability policy for the messaging engine so that the cluster is effectively divided into one set of servers for Version 6 and another set of servers for Version 7.0 or later, and the Version 7.0 or later messaging engine is restricted to the servers at Version 7.0 or later.

Procedure

Make sure that the high availability policy is configured to prevent Version 7.0 or later messaging engines from failing over to messaging engines hosted on a Version 6 server. For information about configuring messaging engines for high availability see “Configuring high availability and workload sharing of service integration” on page 17.

Administering high availability for service integration

Use these tasks to administer high availability at run time.

About this task

- “Managing a messaging engine in a cluster”
- “Moving a messaging engine from one server to another by using the HAManager” on page 26
- “Modifying the failover capability of a messaging engine” on page 26
- “Managing high availability when messaging engines fail to start” on page 26

Managing a messaging engine in a cluster

During run time, you can stop or start a messaging engine in a WebSphere Application Server cluster, independently of the server in which it is running.

About this task

You might want to stop and restart a messaging engine in this way if, for example, you want to take a backup copy.

A stop or start action of a messaging engine applies to the server on which it is performed and is a runtime alteration of the state of the messaging engine. It does not alter the configured “Initial state” of the messaging engine.

If a failover occurs, the fact that you have stopped the messaging engine does not affect the behavior of the newly-activated instance; the messaging engine instance behaves in accordance with the configuration settings of the server in which it is activated.

Moving a messaging engine from one server to another by using the HAManager

You can move a messaging engine from one server to another by changing the policy that is bound to the messaging engine HAGroup. This is the recommended way of moving the messaging engine. You must not attempt to directly activate or deactivate the members of the HAGroup that relates to the messaging engine.

Before you begin

You must stop the messaging engine before changing the policy. After the messaging engine has moved, you have to restart it manually.

About this task

You might want to move a messaging engine if you are aware of a problem that causes the server on which the messaging engine is currently running to fail. For more information, see Specifying a preferred server for messaging requests.

Modifying the failover capability of a messaging engine

You can modify the failover capability of a messaging engine at run time. However, do not do this unless it is absolutely necessary.

About this task

To modify the failover capability of messaging engines, you change the policy that applies to the corresponding high availability group. The servers in the cluster receive the configuration update and the changes take effect as soon as you save them. For example, if there is a “One of N” policy that has no preferred servers, and you change it to add a preferred server and enable fail back, when you click **Save**, the HAManager ensures that the messaging engine is running on the new preferred server; if the messaging engine is running on a different server, the HAManager moves it to the preferred server.

Procedure

1. If you want to change the policy so that the messaging engine runs on a different server, or you want to add new servers, ensure that the servers can access the messaging engine data store or file store.
2. Follow the steps in Selecting the policy for a high availability group to modify the policy.

What to do next

See also High availability manager.

Managing high availability when messaging engines fail to start

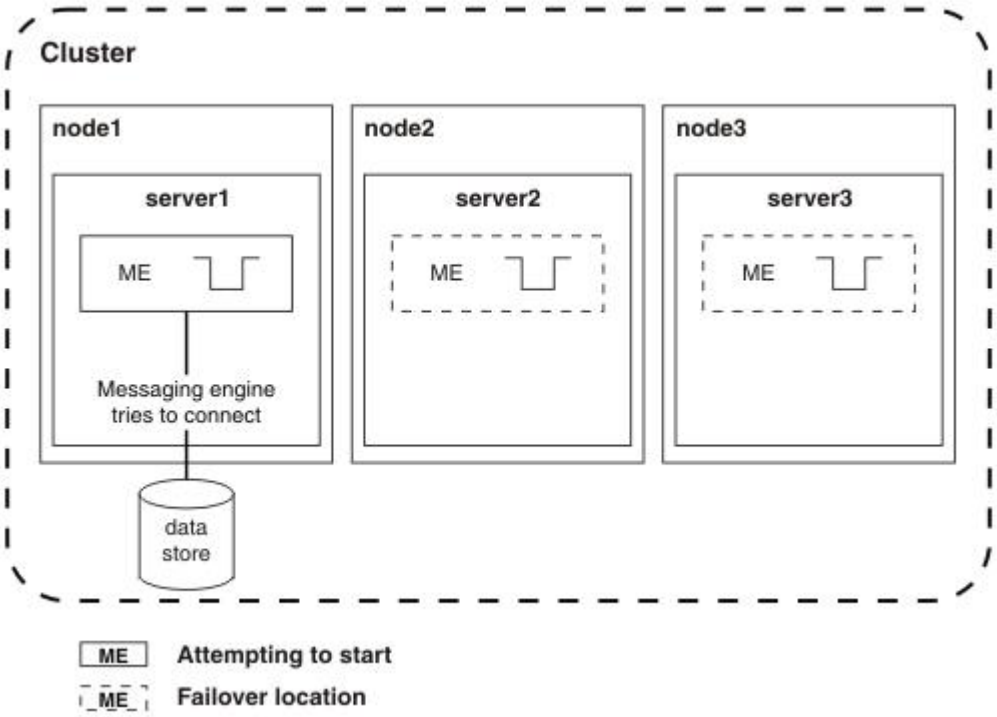
If an attempt to start a messaging engine on a server is unsuccessful, that server is disabled as a location for that messaging engine to run. After you have resolved the problem that prevented the messaging engine from starting, you must manually re-enable the server to maintain your high availability environment.

About this task

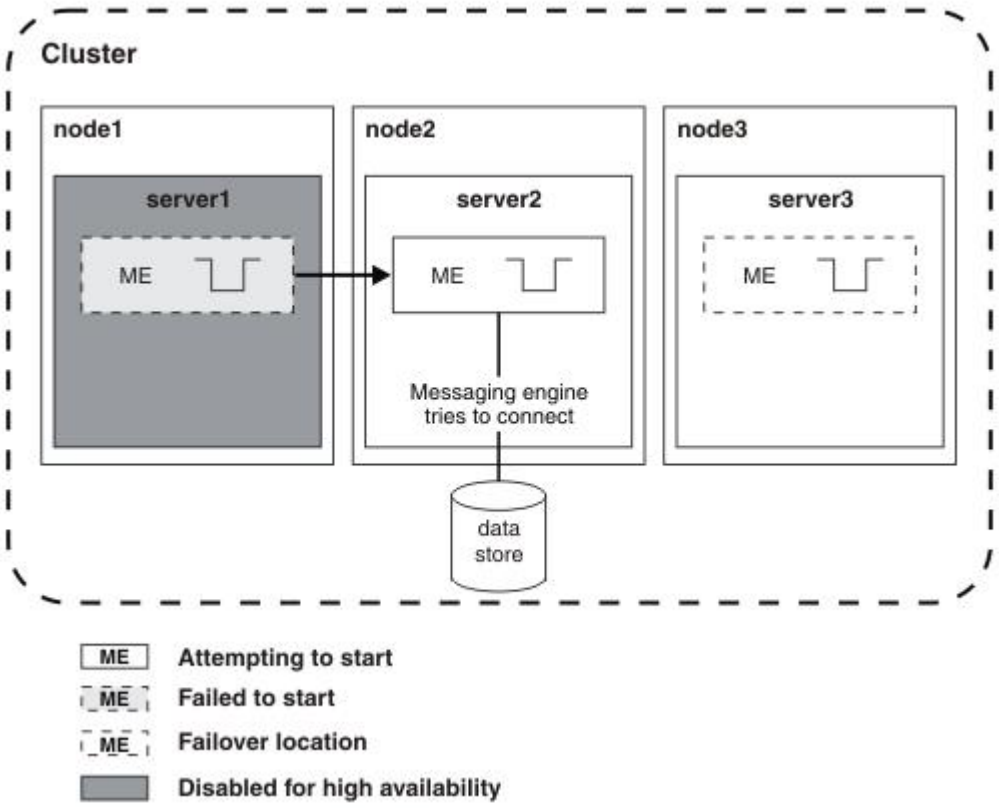
In a high availability environment, a messaging engine can run on multiple application servers. If an attempt to start a messaging engine on a server is unsuccessful, or the server hosting a running messaging engine stops, the high availability manager restarts the messaging engine on another eligible server. If the high availability manager cannot start the messaging engine on that server, the server becomes disabled as a location for that messaging engine to run, and the following message is produced in the JVM logs for that server:

```
CWSID0039E: HAManager-initiated activation has failed, messaging engine messaging_engine_name will be disabled
```

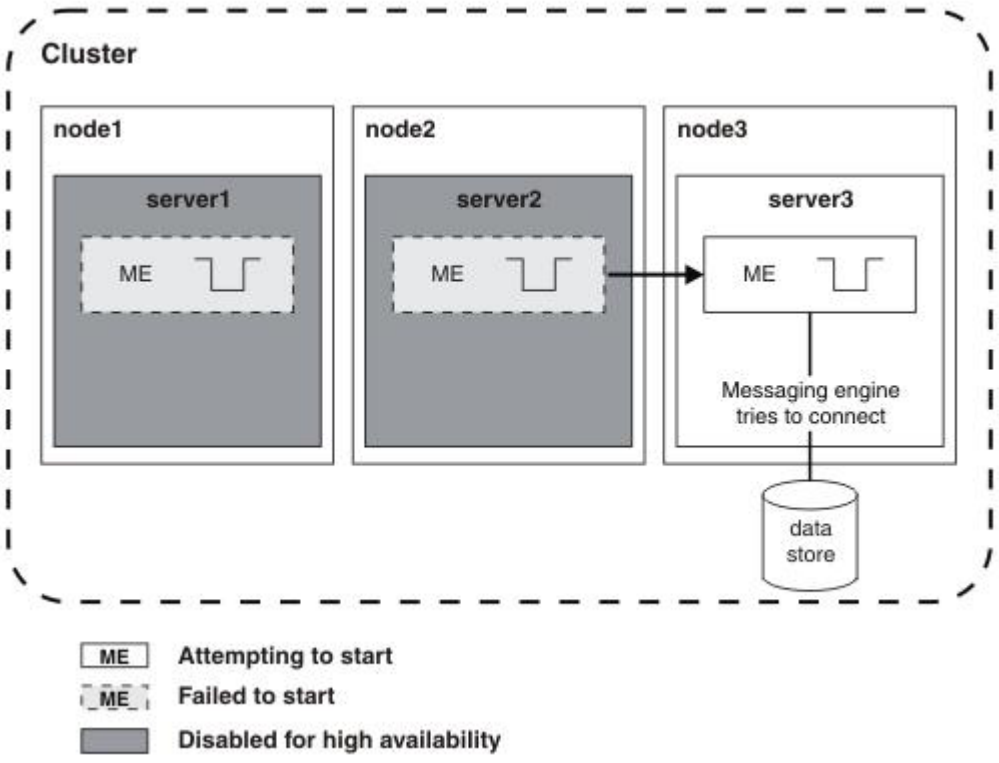
In some situations, the messaging engine can repeatedly fail to start. In the following example, a messaging engine, hosted in a cluster of three servers, is configured to use a data store. The cluster is started before the database that is hosting the data store. The messaging engine attempts to start on server1, and tries to connect to the data store for up to 15 minutes by default.



Because the database has not been started, the messaging engine cannot connect to the data store. The messaging engine fails to start and server1 is disabled for high availability. The messaging engine fails over to server2, and again attempts to start and connect to the data store.



If the database is still not started, the messaging engine fails to start and server2 is disabled for high availability. The messaging engine fails over to server3, and again attempts to start and connect to the data store.



If the database is still not running, the messaging engine fails to start and server3 is disabled for high availability. All servers in the cluster are now disabled for high availability, and the messaging engine cannot start until you start the database and re-enable at least one server.

When you have fixed the cause of the messaging engine's failure to start, re-enable the servers for high availability by either restarting the servers, or by following the steps in this task to enable them using the administrative console.

Procedure

1. Navigate to the high availability groups panel in the administrative console, to display a list of high availability groups. Refer to Viewing high availability group information for details.
2. Find and click the relevant high availability group in the list. To find the relevant group, look for your bus and messaging engine names contained as name-value pairs within the group name. For example the group with the following name contains messaging engine MyCluster.000-MyBus, running on bus MyBus on cluster MyCluster:

   ```
   IBM_hc=MyCluster, WSAF_SIB_BUS=MyBus,WSAF_SIB_MESSAGING_ENGINE=MyCluster.000-MyBus,type=WSAF_SIB
   ```

   The panel for that group appears, showing the high availability state associated with each running server in the messaging engine cluster. If a server is in the disabled state (indicated by a red square), the high availability of your environment is compromised because the messaging engine cannot start on that server. If all servers are in the disabled state, the messaging engine cannot start until you enable at least one server.
3. Select any members that are in the disabled state, and click **Enable**.

What to do next

When a messaging engine that uses a data store fails over to another application server, it might attempt to start before the database server has detected the loss of the network connection to the original application server. Because the database server has not detected the loss of the connection, the data store table locks are not released and the messaging engine cannot start. In this situation, the messaging engine can fail to start on all servers in the cluster. To avoid this problem tune your system to detect the loss of the connection more quickly.

Injecting failures into a high availability system

You can inject failures into the system to check that the high availability behavior functions as you expect.

Before you begin

Attention: This facility is provided to support acceptance testing of a highly available configuration and should only be used for that purpose. Injecting a failure into the system will cause resources to be disabled or failed over from one server to another and will disrupt the workload.

About this task

You can send a JMX command to a messaging engine MBean to simulate a failure in the high availability system. Injecting failures provides a useful way to undertake advanced verification or preproduction testing. You should not inject a failure into a production system.

There are two types of messaging engine failure that you can simulate: local error and global error. For more information about error types, see Messaging engine recovery from exception conditions.

Procedure

1. Start the wsadmin client.

   For more information about the wsadmin client, see wsadmin scripting tool.

2. Use a JMX command to create a variable and set its value to the messaging engine, or engines, that you want to fail.

   In Jython:

```
mbean_name = AdminControl.queryNames("type=SIBMessagingEngine,name=messaging_engine_name,*" )
```

   In Jacl:

```
set mbean_name [$AdminControl queryNames type=SIBMessagingEngine,name=messaging_engine_name,*]
```

3. Use a JMX command to inject the failure, by using the variable you created in the previous step.

   To inject a local error in Jython:

```
AdminControl.invoke(mbean_name, "injectFault", "LocalError")
```

   To inject a global error in Jython:

```
AdminControl.invoke(mbean_name, "injectFault", "GlobalError")
```

   To inject a local error in Jacl:

```
$AdminControl invoke $mbean_name injectFault LocalError
```

   To inject a global error in Jacl:

```
$AdminControl invoke $mbean_name injectFault GlobalError
```

Results

Use the administrative console to view the results. If you have configured the system for failover, a local error should cause the messaging engine to be failed over to another server. A global error does not cause a failover.

Example

For example, to inject a global error into a messaging engine named `myNode01.server1-bus1`, use the following commands:

In Jython:

```
myMBean = AdminControl.queryNames("type=SIBMessagingEngine,name=myNode01.
server1-bus1,*")

$AdminControl invoke $myMBean injectFault GlobalError
```

In Jacl:

```
set myMBean [$AdminControl queryNames type=SIBMessagingEngine,name=myNode01.
server1-bus1,*]

AdminControl.invoke(myMBean, "injectFault", "GlobalError")
```

Chapter 3. Establishing high availability for Transactions

This page provides a starting point for finding information about Java Transaction API (JTA) support. Applications running on the server can use transactions to coordinate multiple updates to resources as one unit of work, such that all or none of the updates are made permanent.

The product provides advanced transactional capabilities to help application developers avoid custom coding. It provides support for the many challenges related to integrating existing software assets with a Java EE environment. More introduction...

Transactional high availability

The high availability of the transaction service enables any server in a cluster to recover the transactional work for any other server in the same cluster. This facility forms part of the overall WebSphere Application Server high availability (HA) strategy.

This feature is in addition to the support for peer restart and recovery, which enables you to restart on a peer system in the sysplex.

As a vital part of providing recovery for transactions, the transaction service logs information about active transactional work in the *transaction recovery log*. The transaction recovery log stores the information in a persistent form, which means that any transactional work in progress at the time of a server failure can be resolved when the server is restarted. This activity is known as *transaction recovery processing*. In addition to completing outstanding transactions, this processing also ensures that any locks held in the associated resource managers are released.

Peer recovery processing

The standard recovery process that is performed when an application server restarts is for the server to retrieve and process the logged transaction information, recover transactional work and complete indoubt transactions. Completion of the transactional work (and hence the release of any database locks held by the transactions) takes place after the server successfully restarts and processes its transaction logs. If the server is slow to recover or requires manual intervention, the transactional work cannot be completed and access to associated databases is disrupted.

To minimize such disruption to transactional work and the associated databases, WebSphere Application Server provides a high availability strategy known as *transaction peer recovery*.

Peer recovery is provided within a server cluster. A peer server (another cluster member) can process the recovery logs of a failed server while the peer continues to manage its own transactional workload. You do not have to wait for the failed server to restart, or start a new application server specifically to recover the failed server.

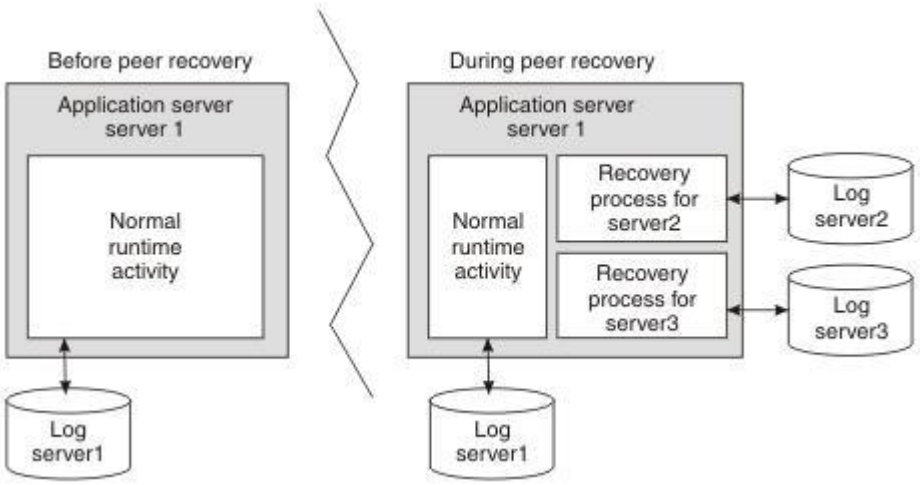


Figure 1. Peer recovery

The peer recovery process is the logical equivalent to restarting the failed server, but does not constitute a complete restart of the failed server within the peer server. The peer recovery process provides an opportunity to complete outstanding work; it cannot start new work beyond recovery processing. No forward processing is possible for the failed server.

Peer recovery moves the high availability requirements away from individual servers and onto the server cluster. After such failures, the management system of the cluster dispatches new work onto the remaining servers; the only difference is the potential drop in overall system throughput. If a server fails, all that is required is to complete work that was active on the failed server and redirect requests to an alternate server.

By default, peer recovery is disabled until you enable failover of transaction log recovery in the cluster configuration, and restart the cluster members. After you enable transaction log recovery, WebSphere Application Server supports two styles for the initiation of transaction peer recovery: automated and manual. You determine which style is more appropriate, based on your deployment, and specify that style by configuring the appropriate high availability policy. This high availability policy is referred to elsewhere in these topics as the *policy for the transaction service*.

Automated peer recovery
: This style is the default for peer recovery initiation. If an application server fails, WebSphere Application Server automatically selects a server to undertake peer recovery processing on its behalf, and passes recovery back to the failed server when it restarts. To use this model, enable transaction log recovery and configure the recovery log location for each cluster member.

Manual peer recovery
: You must explicitly configure this style of peer recovery. If an application server fails, you use the administrative console to select a server to perform recovery processing on its behalf.

In a HA environment, you must configure the compensation logs as well as the transaction logs. For each server in the cluster, use the compensation service settings to configure a unique compensation log location, and ensure that all cluster members can access those compensation logs.

Peer recovery example

The following diagrams illustrate the peer recovery process that takes place if a single server fails. Figure 2 shows three stable servers running in a WebSphere Application Server cluster. The workload is balanced between these servers, which results in locks held by the back-end database on behalf of each server.

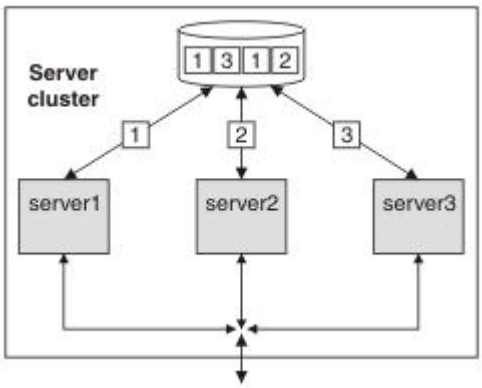


Figure 2. Server cluster up and running, just before server failure

Figure 3 shows the state of the system after server 1 fails without clearing locks from the database. Servers 2 and 3 can run their existing transactions to completion and release existing locks in the back-end database, but further access might be impaired because of the locks still held on behalf of server 1. In practice, some level of access by servers 2 and 3 is still possible, assuming appropriately configured lock granularity, but for this example assume that servers 2 and 3 attempt to access locked records and become blocked.

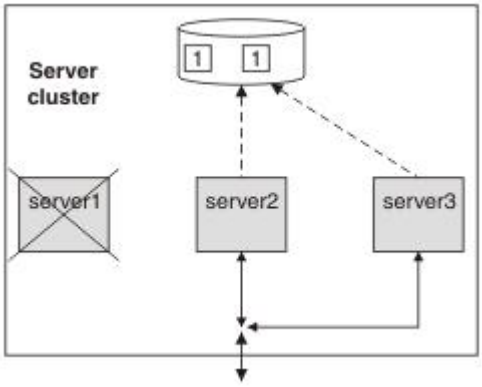


Figure 3. Server 1 fails. Servers 2 and 3 become blocked as a result

Figure 4 shows a peer recovery process for server 1 running inside server 3. The transaction service portion of the recovery process retrieves the information that is stored by server 1, and uses that information to complete any indoubt transactions. In this figure, the peer recovery process is partially complete as some locks are still held by the database on behalf of server 1.

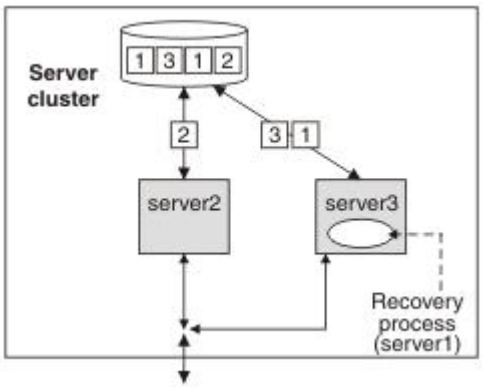


Figure 4. Peer recovery process started in server 3

Figure 5 shows the state of the server cluster when the peer recovery process is complete. The system is in a stable state with just two servers, between which the workload is balanced. Server 1 can be restarted, and will have no recovery processing of its own to perform.

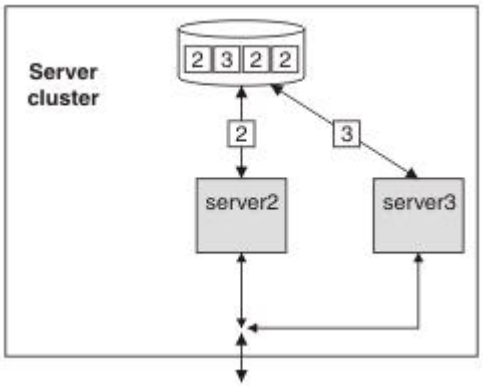


Figure 5. Server cluster stable again with just two servers: server 2 and server 3

Deployment for transactional high availability

Before you use the high availability (HA) function, you must consider deployment issues such as your file system type, or where you plan to store the transaction recovery logs. In particular, your file system type can have important consequences for your recovery configuration.

Common configuration

Transaction peer recovery requires a common configuration of the resource providers between the participating server members to undertake peer recovery between servers. Therefore, peer recovery processing can only take place between members of the same server cluster. Although a cluster can contain servers that are at different versions of WebSphere Application Server, peer recovery can only be performed between servers in the cluster that are at Version 6 or later.

Physical storage

For application servers to perform transaction peer recovery for each other, they must be able to access the transaction recovery logs of all the other members in the cluster. Ensure that the log files are stored on a medium that is accessible by all members of the cluster, and that each cluster member has a unique log file location on this medium. This medium, and access to it, for example through a local area network

(LAN), must support the file-based force operation that is used by the recovery log service to force data to disk. After the force operation is complete, information must be persistently stored on physical disk media.

In a HA environment, application servers must also be able to access the compensation logs. Ensure that the compensation log files are stored on a medium that is accessible by all members of the cluster, and that each cluster member has a unique log file location on this medium.

For example, you can use IBM Network attached storage (NAS) (http://www.ibm.com/servers/storage/nas/index.html) mounted on each node, and shared SCSI drives, but not simple network share. All nodes must have read and write access to the recovery logs.

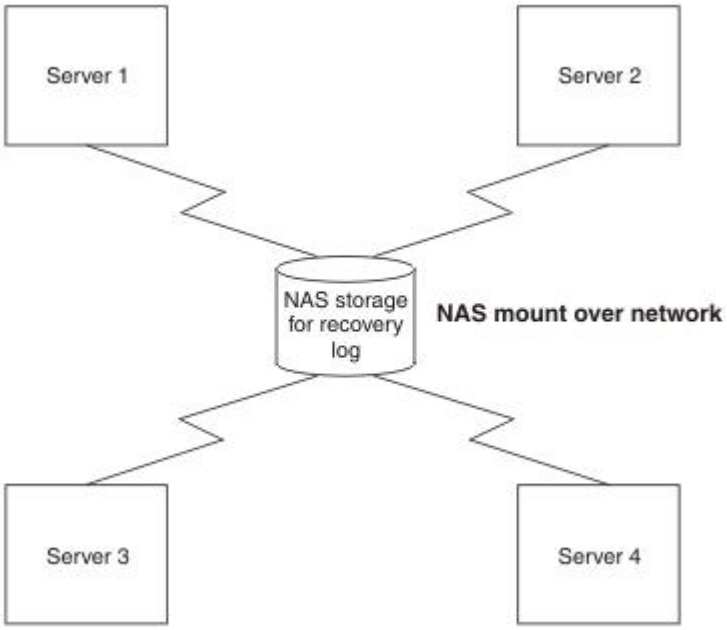


Figure 6. Recovery logs on NAS storage are available to all servers

In addition, configure the mechanism by which the remote log files are accessed, to exploit any fault tolerance in the underlying file system. For example, by using the Network File System (NFS) and hard mounting the remote directory containing the log files by using the -o hard option of the NFS mount command, the NFS client will try a failed operation repeatedly until the NFS server becomes available again.

Two types of potential server failure exist: software failure and hardware failure. Software failures generally do not affect other application servers directly. Even servers on the same physical hardware can undertake peer recovery processing. If a hardware failure occurs, all the servers that are deployed on the failed hardware become unavailable. Servers on other hardware are required to handle peer recovery processing. Any HA configuration requires that servers are deployed across multiple and discrete hardware systems.

File system

The file system type is an important deployment consideration as it is the main factor in deciding whether to use automated or manual peer recovery. For more information, see “How to choose between automated and manual transaction peer recovery” on page 36.

How to choose between automated and manual transaction peer recovery

Your type of file system is the dominant factor in deciding which kind of transaction peer recovery to use. Different file systems have different behaviors, and the file locking behavior in particular is important when choosing between automated and manual peer recovery.

WebSphere Application Server high availability (HA) support uses a heartbeat mechanism to determine whether servers are still running. Servers are considered failed if they stop responding to heartbeat requests. Some scenarios, such as system overloading and network partitioning (explained elsewhere in this topic), can cause servers to stop responding to heartbeats, even though the servers are still running. WebSphere Application Server uses file locking technology to prevent such events from causing concurrent access to transaction recovery logs, because access to a recovery log by more than one server can lead to loss of data integrity.

However, not all file systems provide the necessary file locking semantics, specifically that file locks are released when a server fails. For example, Network File System Version 4 (NFSv4) provides this release behavior, whereas Network File System Version 3 (NFSv3) does not.

NFSv4 releases locks held on behalf of a host in case that host fails. Peer recovery can occur automatically without restarting the failed hardware. Therefore, this version of NFS is better suited for use with automated peer recovery.

NFSv3 holds file locks on behalf of a failed host until that host can restart. In this context, the host is the physical machine running the application server that requested the lock and it is the restart of the host, not the application server, that eventually triggers the locks to release.

To illustrate file locking on NFSv3, consider the behavior when a cluster member fails:

1. Server H is running on host H and holds an exclusive file lock for its own recovery log files.
2. Server P is running on host P and holds an exclusive file lock for its own recovery log files.
3. Host H fails, taking server H with it. The NFS lock manager on the file server holds the locks that are granted to server H on its behalf.
4. A peer recovery event is triggered in server P for server H by WebSphere Application Server.
5. Server P attempts to gain an exclusive file lock for this peer recovery log, but is unable to do so as it is held on behalf of server H. The peer recovery process is blocked.
6. At an unspecified time, host H is restarted. The locks held on its behalf are released.
7. The peer recovery process in server P is unblocked and granted the exclusive file locks that are needed to undertake peer recovery.
8. Peer recovery takes place in server P for server H.
9. Server H is restarted.
10. If peer recovery is still in progress in server P, the recovery is halted.
11. Server P releases the exclusive lock on the recovery logs and returns ownership of the recovery logs back to server H.
12. Server H obtains the exclusive lock and can now undertake standard transaction logging.

Because of this behavior, on NFSv3 you must disable file locking to use automated peer recovery. Disabling file locking can lead to concurrent access to recovery logs so it is vital that you protect your system from system overloading and network partitioning first. Alternatively, you can configure manual peer recovery, where you prevent concurrent access by manually triggering peer recovery processing only for servers that have failed.

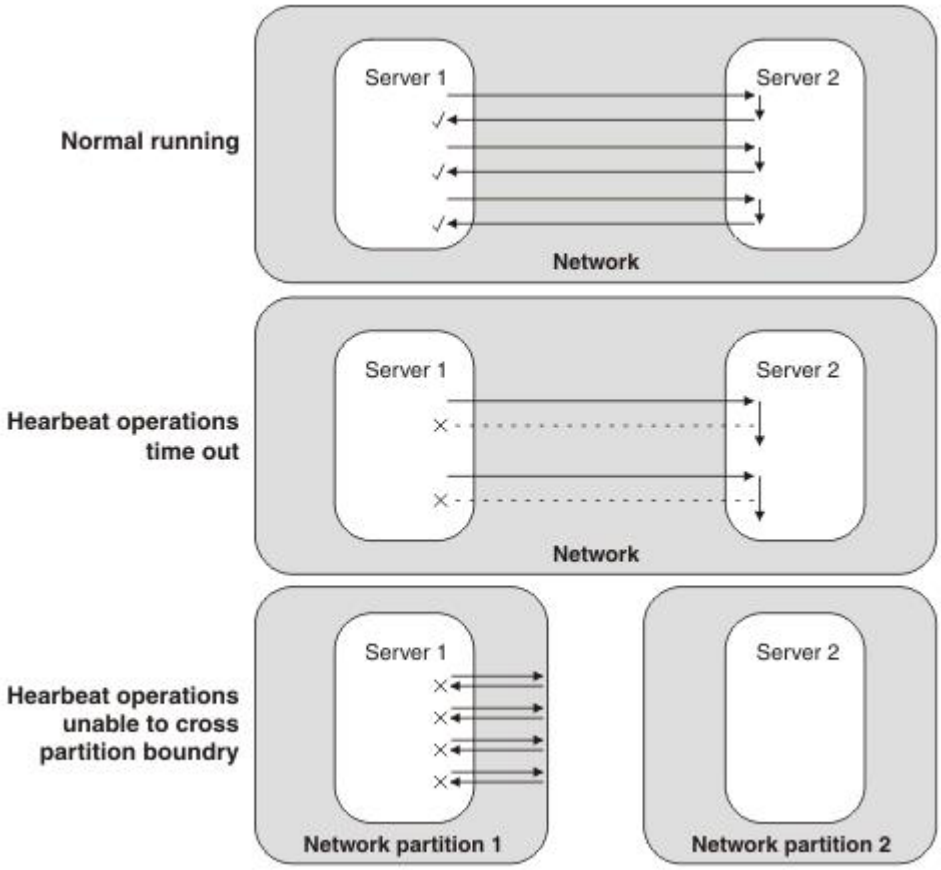
System overloading

System overloading occurs when a machine becomes very heavily loaded such that response times are extremely poor and requests begin to time out. Several potential causes exist for such overloading, including:

- The server is underpowered and cannot handle the workload.
- The server received a temporary surge of requests.
- Insufficient physical memory is available. As a result, the operating system is too busy paging to give the application server the required CPU time.

Network partitioning

Network partitioning occurs when a communications failure in a network results in two smaller networks that are independent and cannot contact each other.



During normal running, two servers on the network exchange heartbeats. During system overloading, heartbeat operations time out, giving the appearance of a server failure. After network partitioning, each server is in a separate network and heartbeats cannot pass between them, also giving the appearance of a server failure.

Figure 7. Heartbeats in a system running normally, compared to heartbeats after the apparent server failures of system overloading and network partitioning

High availability policies for the transaction service

WebSphere Application Server provides integrated high availability (HA) support in which system subcomponents, such as the transaction service, are made highly available. An HA policy provides the logic that governs the manner in which each WebSphere Application Server HA component behaves within the overall HA framework. For the transaction service, the transaction HA policy provides the logic to determine which servers own a recovery log at any time.

Typically, transaction policies assign ownership of a recovery log to the server that originally created it (the home server) and that server can then use the recovery log for both recovery and normal transactional activity. In the event that the home server is unavailable or fails, ownership can pass to a peer server to undertake recovery processing.

Conceptually, a policy can be thought of as consisting of two key components, a policy type and a policy configuration.

Policy type

The policy type determines whether peer recovery initiation is manual or automated. The policy essentially provides the logic for determining updated recovery log ownership in the event of a server failure. The following WebSphere Application Server policy types are used for transaction peer recovery (other HA policy types exist, but are not used by the transaction service):

Static Ownership of the recovery log is defined in the WebSphere Application Server configuration. At run time, the static policy assigns ownership accordingly. Any changes to ownership require a change to the static configuration and therefore this policy type is used for manually initiated peer recovery.

One-of-N
Ownership of the recovery log is determined dynamically by the WebSphere Application Server HA framework and assigned to exactly one of the N cluster members. This policy type is used for automated peer recovery.

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

Default product locations - z/OS

app_server_root
: Refers to the top directory for a WebSphere Application Server node.

  The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are *was.install.root* and *WAS_HOME*.

  The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
: Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

  The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

  The default is /wasv8config/*cell_name*/*node_name*.

plug-ins_root
: Refers to the installation root directory for Web Server Plug-ins.

profile_root
: Refers to the home directory for a particular instantiated WebSphere Application Server profile.

  Corresponding product variables are *server.root* and *user.install.root.*

  In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will always be *app_server_root*/profiles/default because only the profile name "default" is used in WebSphere Application Server for z/OS.

smpe_root
: Refers to the root directory for product code installed with SMP/E or IBM Installation Manager.

  The corresponding product variable is *smpe.install.root*.

  The default is /usr/lpp/zWebSphere/V8R0.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org. You may obtain a copy of the Apache License at http://www.apache.org/licenses/LICENSE-2.0.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Index