This unit describes how to troubleshoot connection pool tuning and management problems in WebSphere® Application Server.

Unit objectives

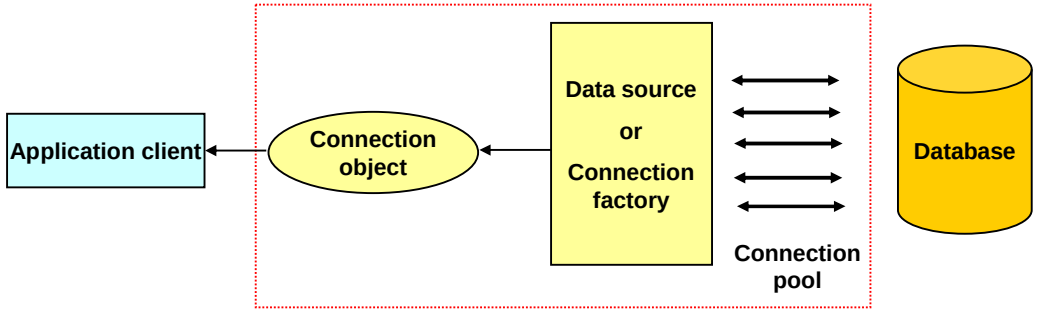
After completing this unit, you should be able to:

- Identify connection pool problems
- Use Tivoli Performance Viewer (TPV) to monitor a connection pool and generate tuning advice
- Enable tracing for connection pool manager components and interpret the trace data
- Perform problem determination tasks to find the root cause of a connection pool problem

After you complete this section, you will be able to identify problems in connection pools, know how to use Tivoli Performance Viewer, TPV, to monitor a connection pool, understand connection pool tracing data, and perform the problem determination tasks to troubleshoot a connection pool problem.

What is connection pooling?

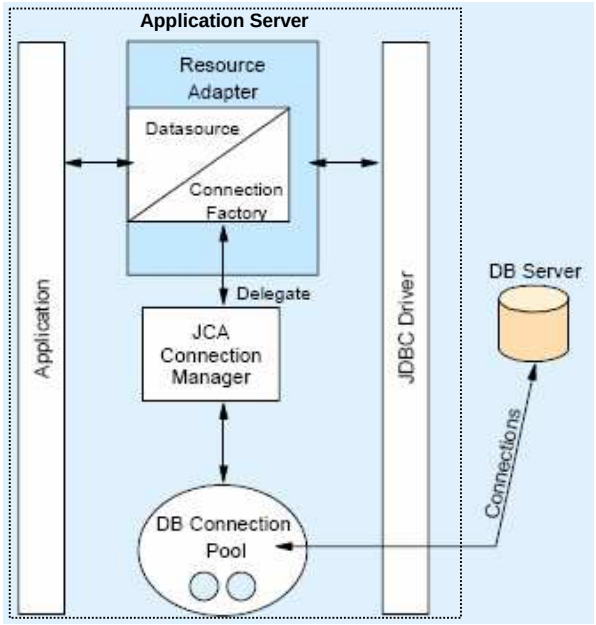
- Application server maintains a pool of *ready-to-use* connections to a data store
- Benefits:
  - Minimizes database session setup and tear-down
  - Improves application database access performance
  - Spreads out connection cost over repeated uses



Applications need to acquire a connection to a data store each time they want to retrieve information from the store. The average connection object is one to two megabytes in size and contains a great deal of information about the connection context. Creating and terminating those connections is actually a very time consuming operation and so it can easily slow down the application. To fix this problem, WebSphere Application Server uses a pool of connections that can be reused by applications. This allows the cost of establishing each connection to be spread out across several requests and can significantly improve performance. An application that needs to access the data store will simply request a connection from the pool and return the connection when it is finished. An example of this work flow is illustated on the slide.

JCA connection pooling architecture

- J2EE Connector Architecture (JCA) V1.5 specification
- Connection pooling is supported by two components:
  - JCA connection manager (called J2C connection pool manager in WebSphere)
  - Relational resource adapter



WebSphere Application Sever implements connection pooling by following the J2EE Connection Architecture version 1.5. There are several objects involved in pooling connections but they can be grouped into two basic components, the JCA connection manager and the Relational Resource Adapter. An application that needs a database connection will go to the Resource Adapter to retrieve a Connection Factory. The Connection Factory will delegate a request to the correct Connection Manager. The Connection Manager is responsible for either returning an existing connection from the pool of available connections or creating a new one if none are available. The application releases the connection when it is finished interacting with the database and the Connection Pool will return it to the pool.

Types of connection pools in WebSphere

- JDBC provider connection pool
  - Enables access to a relational database
  - Provides a connection via a *JDBC™ data source*
    - *Support for WebSphere Version 4 data sources also provided*
  - Managed in the administrative console under *JDBC Providers*
- JMS provider connection pool
  - Enables access to the data store used by the *default messaging engine* or a *WebSphere MQ* provider
  - Provides a connection via a *JMS connection factory*
  - Managed in the administrative console under *JMS Providers*
- EIS Connection pool
  - Enables access to enterprise information systems such as CICS®, legacy databases such as IMS™ and other back-end systems
  - Provides a connection via a *data source* or *connection factory*
  - Managed in the administrative console under *Resource Adapters*

WebSphere Application Server uses a J2C connection pool manager to maintain three different connection pools. The JDBC connection pool is used to manage connections to relational databases such as DB2. This pool can be adjusted by going to the JDBC Providers area of the Administrative Console. There is also a JMS pool for managing requests for connections to a default messaging engine or WebSphere MQ. This pool is maintainted in the JMS Providers section of the adminstrative console. Finally, WebSphere provides an EIS connection pool that manages connections to CICS and legacy back-end systems such as IMS. This connection pool is controled through the administrative console un the Resource Adapters section.

Detecting connection management related problems

- Look in the WebSphere *SystemOut.log* and *SystemErr.log* files for the following types of messages:

Message Prefix or Type	Message Source
DSRA or CWWRA	▪JCA resource adapter
CONM or CWWCM	▪WebSphere Version 4 connection manager ▪Legacy connection manager used to support J2EE 1.2 applications
J2CA or CWWJC	▪J2EE connector (J2C Connection pool manager) ▪Most recent JCA 1.5 compliant connection manager
WSCL or CWWSC	▪WebSphere client (J2EE application client manager)
WTRN or CWWTR	▪WebSphere transaction manager
SQLException or database error code	▪Database manager

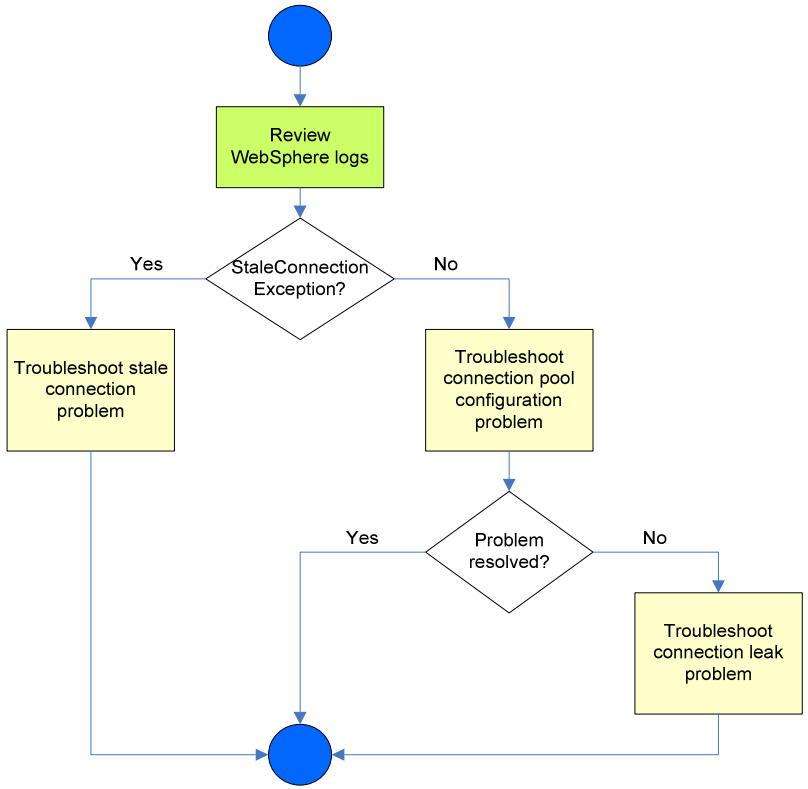
Connection related log messages are sent to the SystemOut and SystemErr logs in the appropriates profile‘s log directory. The System logs are the best starting place to determine if you have a problem in one of the connection pools. WebSphere Application Server maintains connection pools for multiple connection types so it stands to reason that there are several different messages that pertain to the different connection types. The various connection based prefixes are listed on this slide along with the connection types that they pertain to. The best way to determine if you are experiencing connection problems is to search the logs for any of the message prefixes then correlate them to the appropriate message source.

Typical connection pool problem symptoms

- Sporadic failure to connect to an existing data source or connection factory:
- Next, look at the WebSphere logs to see if it additionally shows:
  - No specific exception
    - Probable cause: Improperly tuned connection pool settings
  - ConnectionWaitTimeoutException
    - Probable cause 1: Improperly tuned connection pool settings
    - Probable cause 2: Connection leak
  - StaleConnectionException
    - Probable cause: Stale connection

Most people find out they are experiencing a connection pool problem by noticing symptoms in their application's behavior instead of noticing events in the SystemOut and SystemErr logs. There is usually a problem with a connection pool when an application experiences sporadic failures when trying to connect to a data source. This means the application was able to connect to the data source and work normally but then started to see intermittent failures or a decrease inuser response time. In either case, the next step is to check the log files to help narrow down the prossible cause for the sporadic behavior. There are three possible outcomes from checking the log files. The first outcome is that you do not find connection exceptions. In this case, the problem is likely due to a tuning paramater in the appropriate connection pool. However, if you find ConnectionWaitTimeoutException in the log files then there are two probable causes. You either need to change the connection pool settings or there is a connection leak somewhere in the system. Finally, if you find StaleConnectionException in the log then, as the exception indicates, there is probably a problem with connections going stale.

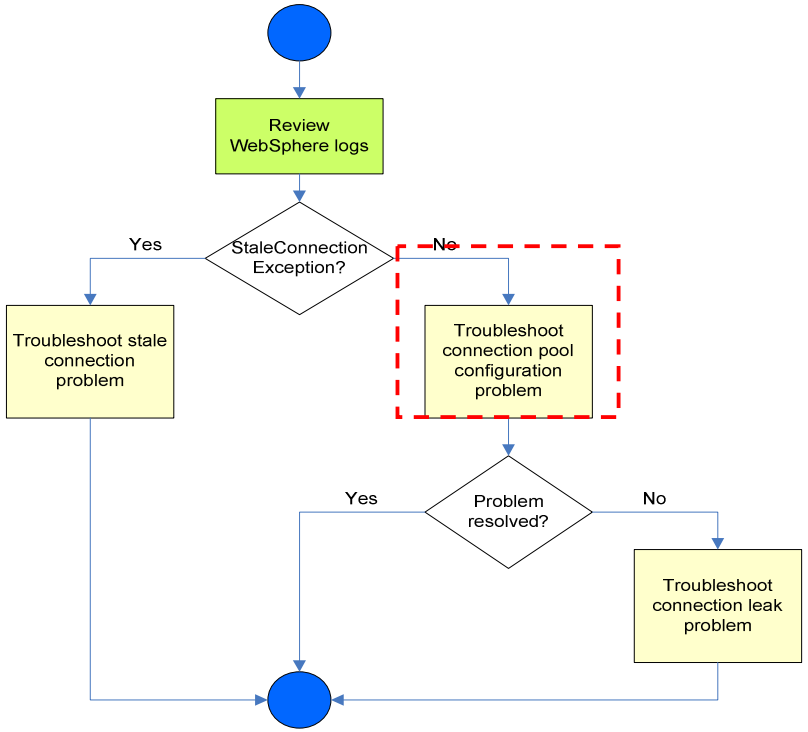
Connection pooling problem determination path



If it helps, you can think of troubleshooting a connection pool problem in terms of a decision tree. The tree starts with the assumption that you are seeing sporatic behavior from your application or you have another reason to believe there is a problem in the connection pool. From there, you review the logs and look for Stale Connection Exceptions. If you find any then the next step is to begin troubleshooting a stale connection problem. Otherwise, it is best to review the connection pool's configuration and make sure it is not causing the problem. If the configuration checks out then you should begin troubleshooting a possible connection leak in the connection pool.

From here, we will take a detailed look at each of the three troublshooting steps.

Troubleshooting connection pool configuration in the problem determination path



In the case where the symptom for a connection pool problem is not accompanied by a StaleConnectionException in the WebSphere logs, start your problem determination effort by looking at the connection pool configuration to rule out any performance tuning issues.

The need for connection pool tuning

- An improperly tuned connection pool can result in:
  - Poor end user response if the client is consistently waiting for a free connection
  - Application exceptions if the client cannot get a connection within the specified wait timeout interval
  - Reduced server throughput if unused connections are wasting system resources
- Connection pools need to be properly tuned to ensure optimal performance:
  - Maximize the chances that connections are available when needed
  - Minimize the number of idle connections
  - Minimize the number of orphaned connections

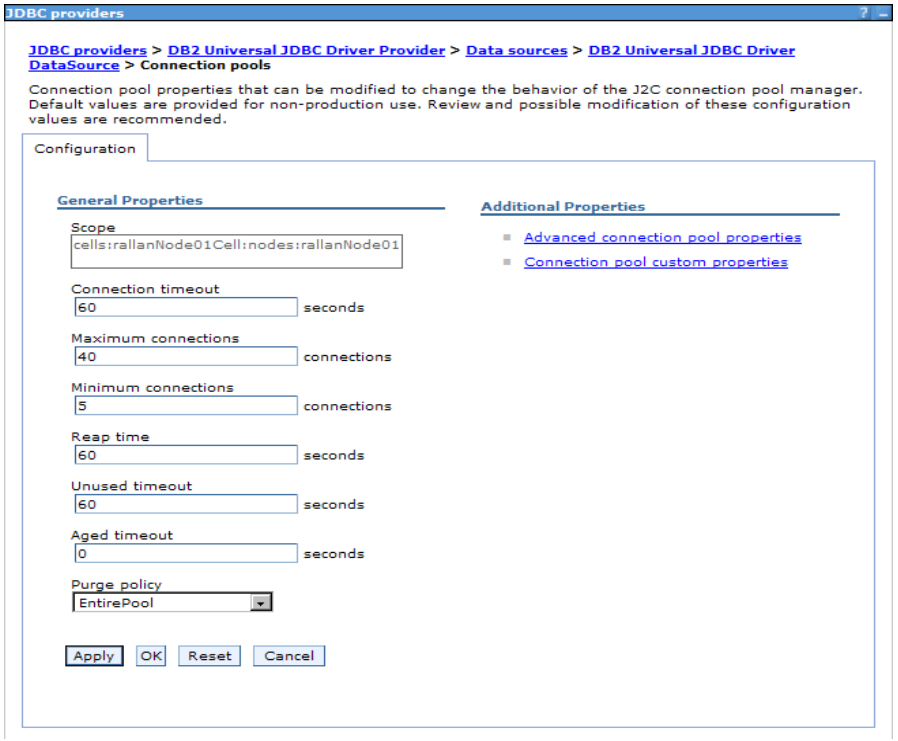
Connection pools allow you to set a range for the number of connections that will maintained by WebSphere Application Server. It is important to get the tuning parameters right otherwise you might inflict the application with problems. Setting the pool size too small can slow down the application because it will have constantly wait for free connectiongs but setting it too large will waste resources and impact the sever's throughput. The timeout can also cause application exceptions if requests go past the time interval. In general, you want to try and tune a connection pool to achieve three goals. First, you want to maximize the change that connections are available when needed. This means setting the connection pool size so that it is big enough to have free connections when they are needed. Second, you want to minimize the number of idle connections because connections that are not being used are overhead that reduces the server's throughput. Finally, you want to set the connection timeout so that it minimizes the number of orphaned connections but does not interfere with connections that are operating normally.

Key connection pool parameters

- Maximum connections
  - Specifies the maximum number of connections that can be created in the pool
  - Default value is 10
  - A value of 0 allows the number of physical connections to grow infinitely and causes the *Connection timeout* value to be ignored
- Connection timeout
  - Specifies the interval, in seconds, after which a connection request times out and a *ConnectionWaitTimeoutException* is thrown.
  - Default value is 180 seconds (three minutes)
  - A value of 0 instructs the pool manager to wait as long as necessary until a connection becomes available

There are a few of the connection pool parameters that play a significant role in achieving the goals we discussed on the previous slide. The first of these paramaters is the maximum connections count. This value governs the maximum size of the connection pool. If the pool has already reached the maximum size it will not allow a new connection to be created and will instead force a request to wait for an existing connection to free up. However, you can set the maximum value to 0 and allow the pool to grow without constraint. This will also cause the Connection Timeout value to be ignored. The connection timeout is how long a connection request will wait for a free connection before it quits and throws a ConnectionWaitTimeoutException. You can also disable the connection timeout by setting it to 0 and allowing a request to wait as long as it takes to receive a connection.

Connection pool parameters in the administrative console

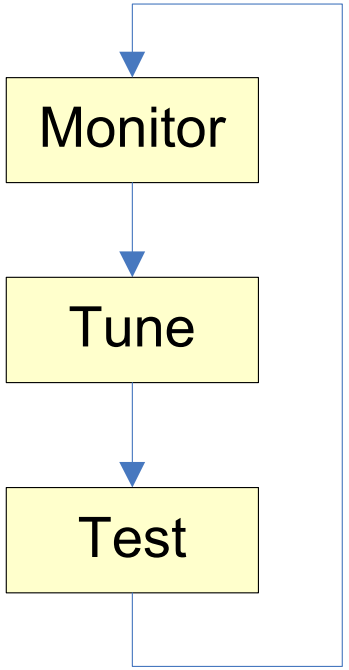


There are several other connection pool properties that can be configured in the administrative console. Minimum Connections, for example, specifies the number of physical connections that should be maintined. Note, this does not mean the connection pool will start with the minimum number of connections but that it will not go beneath that value once it reaches it.

Many of the connection properties interact with eachother. For example, the Reap time specifies, in seconds, the interval between runs of the pool maintenance thread. This value will affect the accuracy of both the Unused timeout and the Aged timeout.

Connection pool tuning tasks

- Monitor connection pool run-time behavior
  - Enable PMI and select desired statistic set
  - View connection pool performance metrics using Tivoli Performance Viewer (TPV)
  - Generate tuning advice using TPV Performance Advisor
- Tune connection pool parameters
  - Make one change at a time
  - Apply recommendations and best practices
- Test application
  - Use a load generation tool to simulate production-like loads
  - Compare results with original baseline
  - Document results



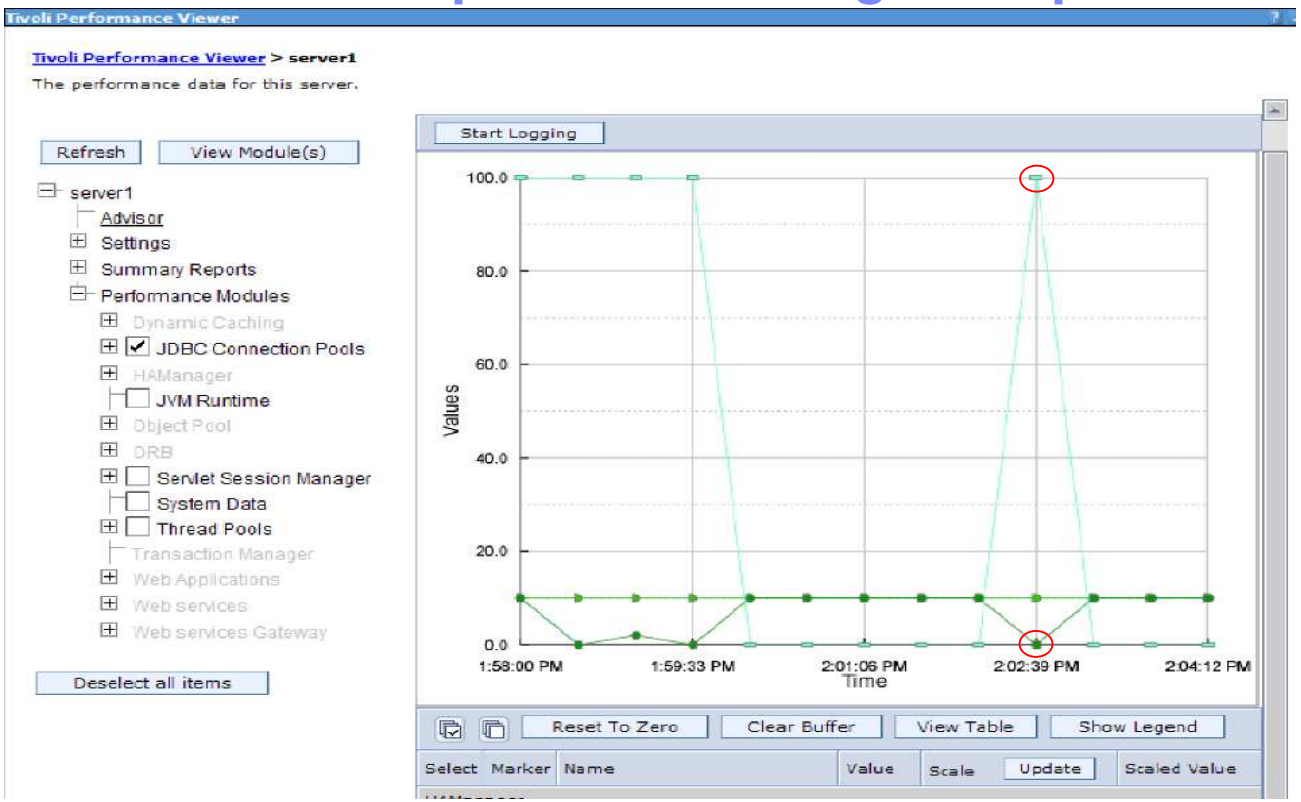
Performance tuning, in general, is an iterative and incremental process consisting of multiple Monitor-Tune-Test cycles. Having the right tools, including a load generation tool to simulate real-world users for load testing, is a must to ensure successful results. The *art* of performance tuning is a mixture of documentation, test data, and experience. There are some tools that can assist with this practice such as the *Tivoli Performance Advisor* embedded in WebSphere Application Server V6, but the suggestions that it offers still need to be verified through load testing. The general method for getting the correct value is to *divide and conquer* by increasing the timeout and connection parameters until the timeout issue disappears and then backing them off until any wasted resources are recovered. Note that the Performance Monitoring Infrastructure (PMI) is enabled by default in WebSphere V6.

Monitoring the connection pool using TPV

Metric Name	Description	What to look for
PoolSize	Size of the connection pool	▪Increases as new connections are created (up to the value of *Maximum connections*) and decreases when connections are destroyed ▪A significant number of creates and destroys is an indication that the pool size (*Maximum connections*) should be adjusted ▪Counter is already enabled as part of the *Basic* (default) PMI statistic set
PercentUsed	Average percent of the pool that is in use	▪If consistently low, you may want to decrease the pool size ▪Counter is already enabled as part of the *Basic* (default) PMI statistic set
WaitingThreadCount	Average number of threads that are concurrently waiting for a connection	▪The optimal value for the pool size is that which reduces this value ▪Counter is already enabled as part of the *Basic* (default) PMI statistic set
PercentMaxed	Average percent of the time that all connections are in use	▪Ensure that you are not consistently maxed at 100% ▪Counter requires the selection of either *All* or *Custom* PMI statistic set

Here are some of the key metrics that WebSphere Application Server monitors in the PMI. These metrics are displayed in TPV under the JDBC Connection Pools and JCA Conection Pools mudules. To access these modules, open the administrative console and natiate to Monitoring and Tuning, then Performance Viewer, then Current Activity. From there, select the server you want to monitor and expand the Performance Modules. Select the appropriate Connection Pool, JDBC ord JCA, and then check the metrics you wish to display.

TPV Connection pool monitoring example



This is a screen camptured example of Tivoli Performance Viewer displaying information on the JDBC Connection Pools.

* The cyan colored graph plots the *PercentUsed* metric (average percent of the pool that is in use).

* The dark green graph plots the *FreePoolSize* metric (number of free connections in the pool).

* The green graph plots the *CreateCount* metric (total number of connections created). 10 connections have been created reaching the default *Maximum connections* value for the pool.

Notice that, as expected, when the *FreePoolSize* is 0 indicating no connection available in the pool, the *PercentUsed* value is at 100%.

Generating tuning advice using TPV Performance Advisor

- TPV Performance Advisor can provide configuration advice for *connection pool size*:
  - Advice appears in the *Performance Advisor* section of TPV
  - Based on collected PMI data over the last one minute interval
  - Uses IBM-defined rules of thumb for advice basis
- Limitations:
  - Pool sizing advice may not be generated if your timeout values are too high (pools are not returning back to minimum values)
  - Advisor only gives recommendations when CPU usage is greater than or equal to 50%

TPV Performance Advisor is one of the ways that WebSphere Application Server can provide tuning advise. TPV Performance Advisor runs on demand and outputs recommendations to a grapical interface in teh administrative console. It's recommendations are based on situations it observes. For example, if it observes that the number of connections is continuously low (equal to the minimum number of connections) then it will recommend that you lower the size of the connection pool. The TPV Performance Advisor can be accessed by opening the administrative console then navigating through Monitoring and Tuning, then Performance Viewer, then finally Current Activity. Here, you select the server you want to monitor and click on Advisor to see the TPV recommendations.

Tuning the connection pool

- Goal is to create a large enough pool that can handle a peak load but does not unnecessarily take up system resources
  - Unused connections during non-peak periods can be controlled with the *Minimum connections* parameter
- In order to successfully tune the connection pool, you need to know two pieces of information:
  - The requests per second that occur during a peak
  - How long the database takes to respond to each type of operation
    - SELECT, INSERT, UPDATE, and so on

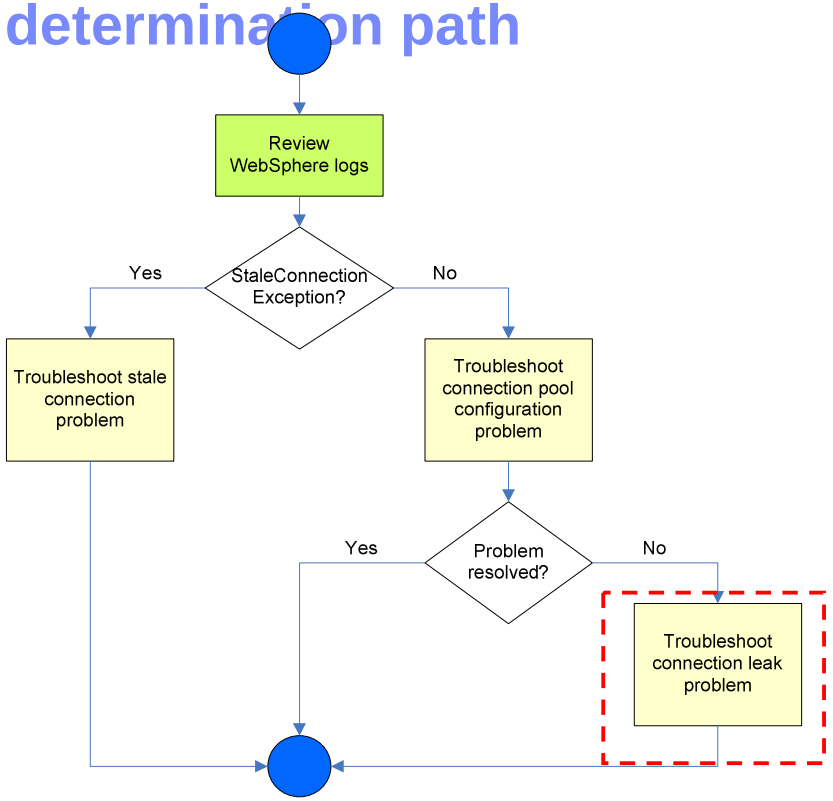
Tuning the connection pool settings for optimal performance during peak load is an iterative activity. The correct parameter values can only be discovered through trial and error. In particular, the two parameters that will have the greatest effect on correcting connection pool configuration errors are the connection timeout and maximum connections. If the time taken to complete a database operation is greater than the amount of time a thread is willing to wait for a resource (the Connection Timeout), then increasing the number of available connections will not solve the problem. Conversely, if the connections are short-lived, then increasing their number could lead to the application server being overloaded in other areas during a peak because the extra connections are unnecessarily consuming resources. Also, the number of idle connections during off peak periods should be weighed against the pool ramp up time when a peak occurs. By understanding the nature of these parameters and the nature of the database operations that will occur during a peak load, an optimal configuration can be achieved leading to optimal performance with the lowest possible overhead.

Connection pool tuning best practices

- Maximum connections setting
  - Double the number of the *Maximum connections* parameter then slowly back it down
  - Better performance is generally achieved if this value is set lower than the value for the maximum size of the Web container thread pool
- Connection timeout setting
  - If a *ConnectionWaitTimeoutException* is found in the WebSphere logs:
    - Obtain the average database operations duration for the application
    - Start with a value that is 5 seconds longer than this average
    - Gradually increase it until problem is resolved or setting is at the highest value that the client will tolerate
- Before you increase the pool size, consult the database administrator
  - Ensure that the database server is configured to handle the maximum pool size setting
  - In a clustered environment, there is the potential of simultaneously allocating *Max connections form all servers simultaneously*

The database connection pool *Minimum* and *Maximum connections* values are often misunderstood. If you set a maximum of 40 connections and a minimum of 10 connections, the pool will not start with 10 connections. The value of 10 connections minimum, is actually a low water mark. Until there are 10 connections required concurrently, the pool will only contain the maximum amount of concurrent connections required up to that point. Therefore, if the number of concurrent connections has only ever reached six, then the pool will contain 6 connections. Once the number of connections needed exceeds 10, the number of connections in the pool will not drop below 10 until the pool is cleaned out. In other words, after the reap time expires, all unused connections will be destroyed until the *Minimum connections* threshold is reached. Configuring a data sources should be done in consultation with the database administrator. For instance, the connection pool size should not be larger than the number of agents or connections allowed on the database server. This can become an issue, especially if cloning is used, because each application server will allocate its own pool. To compute the maximum connections the database may see, multiply the connection pool size by the size of the cluster. For example, assume the connection pool maximum is 10 (the default), and you have a deployment of 2 instances of an application server on each of 2 hosts. There is a potential for up to 40 connections to be open against the database simultaneously. An application that does significantly more INSERT and UPDATE operations than SELECT operations will require greater resources at the database server. If auto indexing is activated then the database could be spending a lot of its time re-indexing after each INSERT or UPDATE. If auto indexing is turned off, then any SELECT operations could become more expensive because the indexes have become stale. In either case, greater overhead will be incurred for applications that are modifying the data in the database.

Troubleshooting connection leaks in the problem determination path



After you have ruled out the possibility of a stale connection and connection pool tuning problem, consider the possiblity of a connection leak.

What is a connection leak?

- A connection leak is a situation that arises when allocated connections are not properly released back to the pool after use.
  - End user response time increase
  - Eventual system lock-up if all worker threads are waiting for a connection
  - *ConnectionWaitTimeoutException*s to be thrown when the *Connection timeout* threshold is reached

A connection leak is typically identified by a *ConnectionWaitTimeoutException* in the WebSphere logs. WebSphere Application Server is smart enough to eventually time-out orphaned connections and return them to the pool, but for an application that makes frequent use of database connections, this might not be enough. New connections can get queued up waiting for the database while old connections are waiting to be timed out. This can bring the application grinding to a halt, and you can see *ConnectionWaitTimeoutExceptions*.

Common causes of connection leaks

- Poorly-written applications often do not properly release database connections
  - Forget to call `connection.close()` in the `finally{}` block
  - Also caused by one method getting a connection, invoking multiple methods, and then forgetting to close the connection when done
- Orphaned connections will only return to the pool after timeout
  - Can cause a back-up of new connections waiting for old connections to time-out
  - New connections that have waited too long throw a *ConnectionWaitTimeoutException*

The most common reason for a connection leak is simply that an application does not deffinsively manage the connections it requests from WebSphere Application Server. This often happens because the application does not properly use the connection.close() method. Connection.close() should be called in the finally{} block to ensure that connections will be closed properly. Unfortunately, connection leaks have traditionally been hard to diagnose because the error messages do not usually provide specific enough information about the source of the problem. A source code review is usually needed to find where the connections are not being properly closed.

Connection leak diagnosis tasks

- Enable connection leak trace facility
  - Use administrative console
- Run and monitor application
  - Wait for *ConnectionWaitTimeoutException*(s) to occur
- Review and analyze trace file
  - Locate source of leak and resolve problem

Determining the root cause of a connection leak will often require a code review. However, there are tools built into WebSphere Application Server that can help narrow down the search. The most useful of which is the connection leak trace facility. Connection leak tracing will allow you to gather more detailed information about the leak and better approach improving the application. The trace utility can help you determine if connections are not being closed or if the application should simply be desigend to use fewer applications.

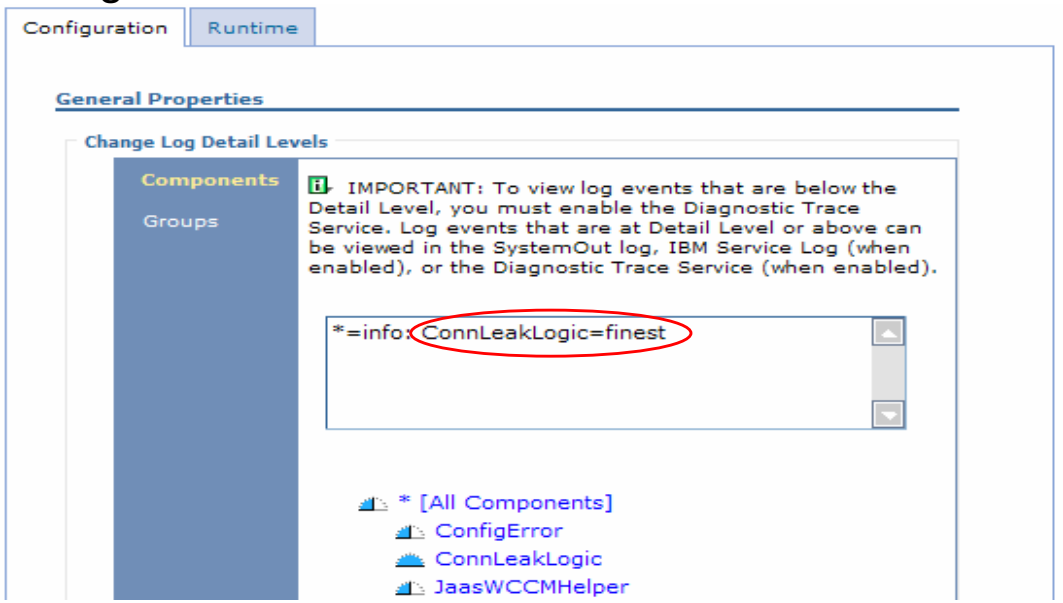
Connection leak trace facility

- A connection leak trace facility is available in WebSphere to provide detailed diagnostic information
  - Prints stack traces of all open connections to *trace.log* when a *ConnectionWaitTimeoutException* occurs
  - Enables you to narrow the search for the responsible source code
  - Light-weight with lower performance overhead than standard connection manager tracing (1-5% impact)
- Limitation:
  - Connection leak trace facility only prints a stack trace of those connections that have been in use for more than 10

When a thread times out waiting on a connection from a full connection pool, it will throw a *ConnectionWaitTimeoutException*. When this exception is thrown, the connection leak tracer will print out the stack traces for every open connection. It does so only when a problem has occurred, providing instant recognition of when it occurred and reduced overhead (1-5%) compared to the WebSphere tracing mechanism. This feature is useful because it shows you the call stacks for all open connections at the time of the exception. This enables you to significantly narrow your search area when you look at the application's source code to try and find the responsible code. It is also be helpful to IBM support, because it will help distinguish between application problems and WebSphere defects. When you enable the connection leak trace facility, for every time interval (the default is 10 seconds), the WebSphere connection pool manager checks how long a connection has been in use and prints the stack trace to the trace.log file. Currently the default time interval is unchangeable. If you have a need to change the default value, contact IBM technical support to obtain an iFix that allows you to add a custom property to the data source configuration.

Enabling the connection leak trace facility

- Enabled using a standard trace string:
  - ConnLeakLogic=finest



The connection leak trace facility is enabled throug the Adminstrative Console. To enable the facility, start to navigating to Troubleshooting, then logs and trace. Select teh application server and then Diagnostic Trace. Make sure loggin is Enabled and then click change log detail levels. You can then specify the desired trace level under the ConnLeakLogic category.

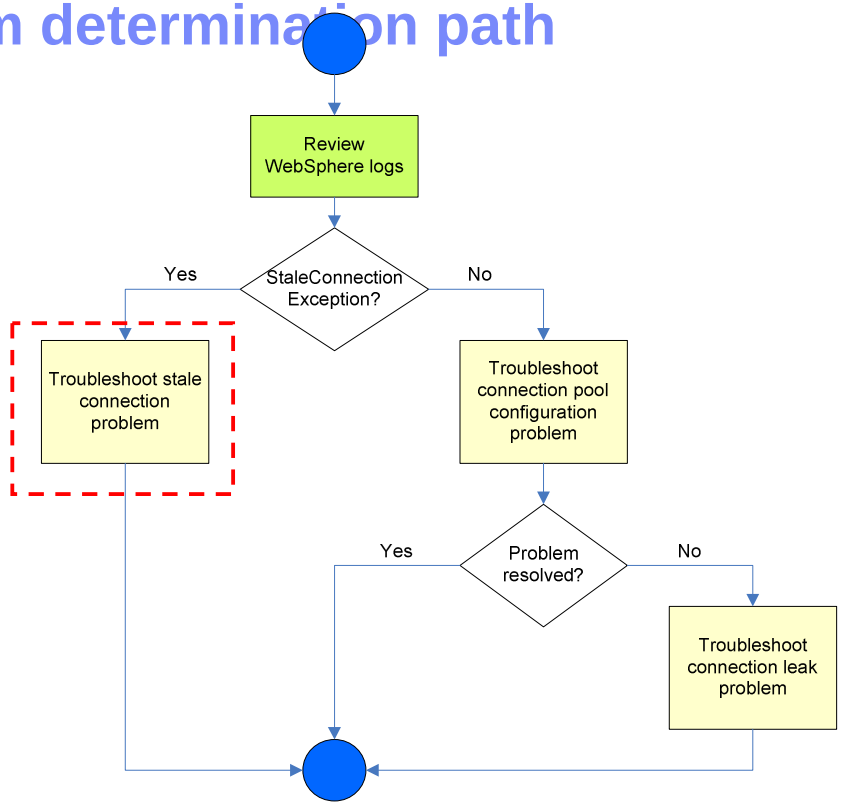
What to look for in the trace

- Search *trace.log* for the string: *Connection Leak Logic Information*
- If present, there are connections that have been in use for more than 10 seconds
- Analyze their stack trace to identify suspect application methods

```
Connection Leak Logic Information:
  MCWrapper id 16089ef2  Managed connection
com.ibm.ws.rsadapter.spi.WSRdbManagedConnectionImpl@6e909ef2
State:STATE_ACTIVE_INUSE Thread Id: 1586lecd Thread Name:
Servlet.Engine.Transports : 1
     Start time inuse Wed Nov 03 08:04:54 CST 2004 Time inuse 20 (seconds)
     Last allocation time Wed Nov 03 08:04:54 CST 2004
       getConnection stack trace information:
  at
com.ibm.ejs.j2c.ConnectionManager.allocateConnection(ConnectionManager.java:565
)
  at
com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:
215)
  at
com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:
306)
  at SnoopServlet.doGet(SnoopServlet.java:130)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:740)
```


There are a few key lines to look for when you start evaluating the trace files. You will first want to look for a line that containts the string Connection Leak Logic Information, followed by a colon. This indicates the start of the connection leak logic output. From there, you should check the time in use and the top of the stack trace for each of the conections. In this example trace, the doGet() method of SnoopServlet has been using a connection for 20 seconds and is therefore a good suspect for a source of a leaking connection.

Troubleshooting stale connections in the problem determination path



If you find StaleConnectionExceptions in the the WebSphere Application Server logs then your choices are clear; start looking for stale connection problems.

What is a stale connection?

- A stale connection problem arises when a connection held by a client is not longer valid.
  - A connection is no longer usable because of a database failure
  - An attempt is made to re-use an orphaned connection (applies only to Version 4.0 data sources)
  - A connection is closed by the Version 4.0 data source auto connection cleanup feature and is no longer usable.

A stale connection is essentially a connection that is held by a client but is no longer a valid connection. One way this can happen is if the other end of the connection, an database for example, experiences a failure and is no longer available. Stale connections can also occur in Version 4.0 data sources when the connection is closed by the connection cleanup feature but the client is still trying to use it. This will happen if the connection has not been used in at least twice the Unused timeout value. At this point, the connection is orphaned and the client will error if it tries to use the connection again.

Recovering from a stale connection

- In general, a stale connection condition indicates that the connection to the database has gone bad.
  - Connection cannot be recovered and must be completely closed rather than returned to the pool.
- Recovering from stale connections is a joint effort between the application server run time and the application developer:
  - Application developer can explicitly catch a stale connection exception and programmatically recover from bad connections (for example, get a new one).
  - Application server will purge the connection pool based on its *PurgePolicy* setting and eliminate the bad connection.

An individual connection can not be recovered once it throws a StaleConnectionException. Instead, the best way to recover from this type of exception is by explicity catching it. Catching a StaleConnectionException while running within the context of a transaction will allow you the avoid having to repeate the entire transaction. One option is to try and complete the pending transaction with a new connection.

It is important to note that the Application server will also take actions to recover from a StaleConnectionException depending on the PurgePolicy setting. It can either clear the entire connection pool, assuming that if one connection went bad then all other connections will likely have the same problem, or just clear the stale conection.

Other stale connection troubleshooting tasks

- Check database or firewall timeout settings
  - Consult with database administrator or network system administrator for the presence of these timeouts
  - If present, they can close connections and cause *StaleConnectionExceptions*
- Determine if a specific query is getting the exception
- Trace the problem using one or more of the following options:
  - WAS.database
  - RRA
  - WAS.j2c

There are several other reasons why a connection might become stale, many of which exist beyond the control of WebSphere Application Server. One common reason is a discrepency between the firewall timeout settings and the connection timeout settings. It is generally a good practice to make sure the connection pool aged timeout is less than the firewall's timeout and that both are less than the database timeout. It is also possible that you are experiencing a StaleConnectionException becasue the returned SQLCode maps to a StaleConnection. If you aren't able to find the source of the problem by taking a quick look at the various components involved in the connection then your best bet is to turn on tracing and gather more information. This is can be very useful when a connection is unusable because of a SQLException that did not imediately map to a StaleConnectionException but eventually resulted in one being thrown.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject= Feedback about SW5706G09_ConnectionPool.ppt

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

CICS IMS Perform WebSphere

J2EE, JDBC, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.