IBM Software Group

Understanding and Resolving ConnectionWaitTimeoutExceptions in WebSphere Application Server

Hobert Bush - bushh@us.ibm.com
David Tiler - dtiler@us.ibm.com
WebSphere Application Server L2 Support Analysts
September 18, 2013

WebSphere® Support Technical Exchange

Agenda

- Connection Pooling Overview
- Connection Wait Timeouts
- Problem Symptoms
- Data Collection
- Data Analysis
  - Five common root causes
  - How to resolve problem for each root cause
- Questions and Answers

Connection Pooling Overview

- Creating a database connection is an expensive operation
- WebSphere pools connections to improve performance
- Connections are returned to the pool when closed by the application and reused
- Connection pooling implementation is based on the Java Connector Architecture (JCA) specification, also known as J2C
  - V5.0 - V5.1: JCA 1.0
  - V6.0 - V7.0: JCA 1.5
  - V8.0 - V8.5: JCA 1.6
- Supports connections to any backend if a resource adapter is provided
- Resource adapter for databases ships with WAS – it is the “Relational Resource Adapter” (RRA)
- Resource Adapters for JMS also ship with WAS, other resource adapters are supplied by vendors
- To use database connection pool, a WebSphere data source must be created

Connection Wait Timeouts

- Connection pool has a maximum size; default value is 10
- Connection pool can create up to the maximum number of connections and allocate them to applications
- To get a connection, application must perform JNDI lookup of data source, and call getConnection() on the data source
- WebSphere will return a connection in the free pool (not being used by another thread) if available
- WebSphere will create a new connection if there are no connections in free pool and connection pool is below its maximum size
- If pool is at its maximum size, and all connections are in use (none in free pool), thread will wait for connection
- Thread will wait for the Connection Timeout number of seconds (default is 180 seconds)
- ConnectionWaitTimeoutException thrown after Connection Timeout expires no connection can be allocated

Problem Symptoms

- Several problem symptoms may occur:

1. J2CA0045E error
   J2CA0045E: Connection not available while invoking method createOrWaitForConnection for resource jdbc/db2demo.
2. ConnectionWaitTimeoutException
   com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException: Connection not available, Timed out waiting for 180004
3. Slow performance
   Application running slower than expected, and transaction timeouts (WTRN0006W) could occur
4. Threads “hung” waiting for free connection...

Threads “hung” waiting for free connection

- Javacore or thread dump shows many threads with the following at top of the stack:

```
3XMTHREADINFO      "WebContainer : 6" J9VMThread:0x3C328200, j9thread_t:0x3A7289EC,
java/lang/Thread:0x944DEC48, state:CW, prio=5
3XMTHREADINFO1           (native thread ID:0x46B0DB, native priority:0x5, native policy:UNKNOWN)
3XMTHREADINFO3          Java callstack:
4XESTACKTRACE              at java/lang/Object.wait(Native Method)
4XESTACKTRACE              at java/lang/Object.wait(Object.java:196(Compiled Code))
4XESTACKTRACE              at com/ibm/ejs/j2c/FreePool.queueRequest(FreePool.java:354(Compiled Code))
4XESTACKTRACE              at com/ibm/ejs/j2c/FreePool.createOrWaitForConnection(FreePool.java:1274(Compiled Code)
)
4XESTACKTRACE              at com/ibm/ejs/j2c/PoolManager.reserve(PoolManager.java:2524(Compiled Code))
4XESTACKTRACE              at
com/ibm/ejs/j2c/ConnectionManager.allocateMCWrapper(ConnectionManager.java:1059(Compiled Code))
4XESTACKTRACE              at
com/ibm/ejs/j2c/ConnectionManager.allocateConnection(ConnectionManager.java:696(Compiled Code))
4XESTACKTRACE              at
com/ibm/ws/rsadapter/jdbc/WSJdbcDataSource.getConnection(WSJdbcDataSource.java:668(Compiled Code))
```

- WSV0605W messages may appear in SystemOut.log with similar stack trace

```
ThreadMonitor W   WSVR0605W: Thread "WebContainer : 43" (0000003f) has been active for 670186 milliseconds and
may be hung.  There is/are 40 thread(s) in total in the server that may be hung.
```

Data Collection - Tracing

- Enable connection pool trace
  - *=info:WAS.j2c=all:RRA=all
  - Best option if problem is easily reproducible
  - Better to enable on Configuration tab in trace settings to get trace from startup
  - With Runtime trace, data is only captured after trace is enabled
    - Time that connection was created and allocated might be missing
    - Trace entries showing stack trace of thread that called getConnection() might not be generated

Data Collection – showPoolContents

- showPoolContents command outputs state of connection pool
  - This is the same information that can be found in trace
  - Often contains enough to determine cause of connection wait timeouts
- To invoke from command line:

```
wsadmin>set ds [$AdminControl queryNames "*:name=<DISPLAY NAME OF DATASOURCE>,process=<SERVER NAME>,node=<NODE NAME>,j2eeType=JDBCDataSource,*"]
wsadmin>$AdminControl invoke $ds showPoolContents
```

- Can also be invoked from application using JMX API
- Lightweight ConnLeakLogic=all trace can be enabled to show stack traces of getConnection() calls where the connection is used for more than 10 seconds
  - Does not write anything to trace unless other traces enabled
  - Output will be included in showPoolContents output

How to read trace data

- Identify name of connection pool, and search for "PoolManager name:jdbc/db2demo" (where jdbc/db2demo is pool name)
- You will find output in same format as showPoolContents output
- Find current and maximum connection pool size:
  Total number of connections: 5 (max/min 5/0, reap/unused/aged 180/1800/0, connectiontimeout/purge 180/EntirePool)
- For connections in use, check number of connections in “Shared Connection information” and “UnShared Connection information”
- Identify type of transaction (global transaction or LTC):
  com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5a69030
  com.ibm.ws.tx.jta.TransactionImpl@6cb16cb1#tid=6
- “MCWrapper id” is a unique identifier for each connection
- Check “Thread Id” and “Thread Name”
  - Should be different for each connection
  - Thread Name can be found in Javacore or thread dump
- Handle count is the number of connection handles that have been allocated but not closed
  - Value of 0 means application has called close()
  - Value greater than 1 means app getting multiple connections

Possible root causes

- Use of Shareable Connections in Long-Running LTC
- Multiple connection requests on same thread without calling close()
- Connection leak in application
- Long-running queries or slow database response time
- Maximum Connections set too low

What is a LTC?

- LTC is Local Transaction Containment, started every time a servlet is invoked, or EJB without container-managed transactions
- Started in the absence of a global transaction
- Contains one or more Resource Manager Local Transactions (RMLTs)
- LTC is not controlled by the application
- Shareable connections will not return to free pool until LTC ends
  - Remain allocated for possible reuse within the LTC
- LTC ends when servlet service method or EJB method exits
- Long-running LTCs can cause connections to be held too long
- Connections are automatically closed when the LTC ends

Root Cause #1- Shareable Connections in Long-Running LTC

[9/4/13 0:13:56:046 EDT] 0000003a PoolManager 3 reserve()
PoolManager name:**jdbc/db2demo**
PoolManager object:93385150
Total number of connections: 5 (max/min 5/0, reap/unused/aged 180/1800/0, connectiontimeout/purge 180/EntirePool)
(testConnection/inteval false/0, stuck timer/time/threshold 0/0/0, surge time/connections 0/-1)

Shared Connection information (shared partitions 200)
com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5a69030;RUNNING; MCWrapper id 5874c60 Managed connection WSRdbManagedConnectionImpl@5851fb1 State:STATE_TRAN_WRAPPER_INUSE **Thread Id: 00000039 Thread Name: WebContainer : 6 Handle count 0**
com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5a5b88d;RUNNING; MCWrapper id 587e8c2 Managed connection WSRdbManagedConnectionImpl@584c204 State:STATE_TRAN_WRAPPER_INUSE Thread Id: 00000035 Thread Name: WebContainer : 2 Handle count 0
com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5a5c7b8;RUNNING; MCWrapper id 58818f5 Managed connection WSRdbManagedConnectionImpl@58691f7 State:STATE_TRAN_WRAPPER_INUSE Thread Id: 00000037 Thread Name: WebContainer : 4 Handle count 0
com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5a65226;RUNNING; MCWrapper id 588d87d Managed connection WSRdbManagedConnectionImpl@5889e58 State:STATE_TRAN_WRAPPER_INUSE Thread Id: 00000036 Thread Name: WebContainer : 3 Handle count 0
com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5a5d743;RUNNING; MCWrapper id 58897ba Managed connection WSRdbManagedConnectionImpl@587197a State:STATE_TRAN_WRAPPER_INUSE Thread Id: 00000033 Thread Name: WebContainer : 0 Handle count 0
Total number of connection in shared pool: 5
Free Connection information (free distribution table/partitions 5/1)

No free connections

UnShared Connection information
No unshared connections

Root Cause #1- Shareable Connections in Long-Running LTC

- The trace data shows shareable connection in a LTC.
- The application requested the connection to be closed as denoted by a Handle count of 0
- The connection will remain allocated in the pool until the LTC ends.

Shared Connection information (shared partitions 200)
com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5a69030;RUNNING; MCWrapper id 5874c60 Managed connection WSRdbManagedConnectionImpl@5851fb1 State:STATE_TRAN_WRAPPER_INUSE **Thread Id: 00000039 Thread Name: WebContainer : 6 Handle count 0**

- When the application closes a shareable connection, the connection is not truly closed, nor is it returned to the Free pool. Rather, it remains in the Shared connection pool, ready for another request within the same LTC for a connection to the same resource.

How to resolve Root Cause #1

- Connection usage can be encapsulated in a global transaction (UserTransaction). The connection will return to the free pool when the global transaction commits
- Application can be changed so that the time between connection.close() and LTC end is reduced
- The simplest solution is to use unshareable connections, rather than shareable. This will allow connections to be removed from the pool immediately once the application calls close()
  - Application Change:
    - Resource References
      - Res-sharing-scope: Unshareable
  - Connection Pool Custom Property:
    - globalConnectionTypeOverride = unshared
      - Will override all other settings, including resource references
    - defaultConnectionTypeOverride = unshared
      - Will provide the default if none specified on resource reference

Shareable vs. Unshareable Connections

- With unshareable connections, only one connection handle to the physical connection is created
- With shareable connections, one connection handle is created for each getConnection() call within a transaction (Global or LTC)
- Default is shareable, as mandated by JCA specification (Section 7.9 for JCA 1.6)
- Functionality is effectively the same if the application gets only one connection within a transaction
- If application gets multiple connections in one transaction, shareable connections will result in the same connection being used (multiple handles to the same physical connection)
  - Application must close() first connection before getting second connection (get – use – close pattern)
- Performance impact is negligible because with unshareable connections, a second getConnection() call would likely get a connection from the free pool (avoiding overhead of new connection)

Root Cause #2 - Multiple connection requests without calling close()

[9/3/13 18:32:07:917 EDT] 00000003 PoolManager 3 alarm(), Pool contents ==> PoolManager name:**jdbc/db2demo**
PoolManager object:92444554

Total number of connections:5 (max/min 5/0, reap/unused/aged 180/1800/0, connectiontimeout/purge 180/EntirePool)
(testConnection/inteval false/0, stuck timer/time/threshold 0/0/0, surge time/connections 0/-1)
(pool paused false, prePopulate alternate false, resourceFailBackEnabled true, isAlternateResourceEnabled false, disableDatasourceFailoverAlarm false, startFailBack false)
(isPartialResourceAdapterFailoverSupportEnabled false, isAlteranteResourcePoolManager false, resourceAvailabilityTestRetryInterval 10, currentInusePool null, currentMode 100, alternate jndiName null)

The waiter count is 7
The mcWrappers in waiter queue []
Shared Connection information (shared partitions 200)

com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5bdc62c;RUNNING; MCWrapper id 59c201d
Managed connection WSRdbManagedConnectionImpl@59bc690 State:STATE_TRAN_WRAPPER_INUSE **Thread Id: 0000006a** Thread Name: WebContainer : 0 **Handle count 1** Start time inuse Tue Sep 03 18:26:07 EDT 2013 Time inuse 360 (seconds)

com.ibm.ws.LocalTransaction.LocalTranCoordImpl@5bdc62c;RUNNING; MCWrapper id 5b71be1
Managed connection WSRdbManagedConnectionImpl@5b6dff2 State:STATE_TRAN_WRAPPER_INUSE **Thread Id: 0000006a** Thread Name: WebContainer : 0 **Handle count 1** Start time inuse Tue Sep 03 18:26:08 EDT 2013 Time inuse 359 (seconds)

How to Resolve Root Cause #2

- Trace data shows different shared connections **(MCWrapper id 5b71be1 and MCWrapper id 59c201d)** are running on the same thread **(Thread Id: 0000006a)** with a **Handle count 1** (not Root Cause #1)
  - In this case, these are nested connections in the same thread.
  - Each created connection is dependent on all the other connections allocated on the same thread. Thus, the connection must be closed before the transaction can end and the connections are removed from the pool.
  - If the application issued a close on connection **MCWrapper id 5b71be1**, it would remain allocated until the LTC end or the global transaction was committed, and the same connection would be used for the second getConnection() call
- Application needs to follow get – use – close pattern, and close() first connection before subsequent getConnection() calls
- Nested connections should be avoided if possible. If the application code cannot be changed, switch to unshareable connections and increase maximum connection pool size to work around this problem.

Root Cause #3 – Connection Leak in Application

[9/4/13 8:38:30:570 EDT] 00000003 PoolManager 3 alarm(), Pool contents ==> PoolManager name:**jdbc/db2demo**
PoolManager object:89943098
Total number of connections: 5 (max/min 5/0, reap/unused/aged 180/1800/0, connectiontimeout/purge 180/EntirePool)
(testConnection/inteval false/0, stuck timer/time/threshold 0/0/0, surge time/connections 0/-1)
Shared Connection information (shared partitions 200)
No shared connections

Free Connection information (free distribution table/partitions 5/1)

No free connections

UnShared Connection information
MCWrapper id 5c11f3e Managed connection WSRdbManagedConnectionImpl@5bee0c0 State:STATE_ACTIVE_INUSE Thread Id: 00000024 Thread Name: WebContainer : 5 **Handle count 1 Start time inuse Wed Sep 04 08:35:32 EDT 2013 Time inuse 178 (seconds)**
MCWrapper id 5c140f6 Managed connection WSRdbManagedConnectionImpl@5c085b4 State:STATE_ACTIVE_INUSE Thread Id: 00000025 Thread Name: WebContainer : 6 Handle count 1 Start time inuse Wed Sep 04 08:35:32 EDT 2013 Time inuse 178 (seconds)
MCWrapper id 5bf5554 Managed connection WSRdbManagedConnectionImpl@5bf19dd State:STATE_ACTIVE_INUSE Thread Id: 00000022 Thread Name: WebContainer : 3 Handle count 1 Start time inuse Wed Sep 04 08:35:32 EDT 2013 Time inuse 178 (seconds)
MCWrapper id 5ba9a36 Managed connection WSRdbManagedConnectionImpl@5ba417e State:STATE_ACTIVE_INUSE Thread Id: 00000021 Thread Name: WebContainer : 2 Handle count 1 Start time inuse Wed Sep 04 08:35:30 EDT 2013 Time inuse 180 (seconds)
MCWrapper id 5c16e0c Managed connection WSRdbManagedConnectionImpl@5c0b8ff State:STATE_ACTIVE_INUSE Thread Id: 0000001f Thread Name: WebContainer : 0 Handle count 1 Start time inuse Wed Sep 04 08:35:32 EDT 2013 Time inuse 178 (seconds)
Total number of connection in unshared pool: 5

Connection Leak Logic Information:
MCWrapper id 5c11f3e Managed connection WSRdbManagedConnectionImpl@5bee0c0 State:STATE_ACTIVE_INUSE Thread Id: 00000024 Thread Name: WebContainer : 5 **Handle count 1**
Start time inuse Wed Sep 04 08:35:32 EDT 2013 Time inuse 178 (seconds)
Last allocation time Wed Sep 04 08:35:32 EDT 2013
getConnection stack trace information:
com.ibm.ejs.j2c.ConnectionManager.allocateConnection(ConnectionManager.java:1260)
com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:669)
com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:636)
com.ibm.wste.WSTEServlet.doPost(WSTEServlet.java:62)

How to resolve Root Cause #3

- Trace Data shows unshared connections in a LTC or a shared connection enlisted in a global transaction (not Root Cause #1)
- Each connection is allocated on a different thread (not Root Cause #2)

Connection Leak Logic Information:
MCWrapper id 5c11f3e Managed connection WSRdbManagedConnectionImpl@5bee0c0 State:STATE_ACTIVE_INUSE Thread Id: 00000024 Thread Name: WebContainer : 5 **Handle count 1**
Start time inuse Wed Sep 04 08:35:32 EDT 2013 Time inuse 178 (seconds)
Last allocation time Wed Sep 04 08:35:32 EDT 2013
getConnection stack trace information:
com.ibm.ejs.j2c.ConnectionManager.allocateConnection(ConnectionManager.java:1260)
com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:669)
com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:636)
com.ibm.wste.WSTEServlet.doPost(WSTEServlet.java:62)

- The application is not calling close() or not calling close() properly on the connections based on the Handle count being 1
- The Connection Leak Logic information outputs the code making the getConnection() call if leaking is suspected after ten seconds.
- Review the application and ensure the referenced getConnection() call has a matching close() call.

```
61        ds = (DataSource) ctx.lookup("jdbc/db2demo");
62        conn = ds.getConnection();
63        pStmt = conn.prepareStatement("Select EMPNO, FIRSTNME, LASTNAME FROM DB2INST1.EMPLOYEE");
64        rs = pStmt.executeQuery();
```

How to properly close connection – finally block

```
DataSource ds = null;
Connection conn = null;
PreparedStatement pStmt = null;
ResultSet rs = null;

try {
  InitialContext ctx = new InitialContext();
  ds = (DataSource) ctx.lookup("jdbc/db2demo");
  conn = ds.getConnection();

  pStmt = conn.prepareStatement("select * from employee");
  rs = pStmt.executeQuery();

  while(rs.next())
    out.println(rs.getString(1));

  rs.close();
  pStmt.close();
  conn.close();
}

catch (Exception e) {
  e.printStackTrace();
}

finally {
  rs.close();
  pStmt.close();
  conn.close();
}
```

Root Cause #4 – Long running queries

- Long-running queries or slow database response time

[9/3/13 21:29:41:393 EDT] 00000028 PoolManager 3 reserve() entry, Pool contents ==> PoolManager name:jdbc/db2demo
PoolManager object:1581276736
Total number of connections: 5 (max/min 5/0, reap/unused/aged 180/1800/0, connectiontimeout/purge 180/EntirePool)
(testConnection/inteval false/0, stuck timer/time/threshold 0/0/0, surge time/connections 0/-1)
The waiter count is 10
The mcWrappers in waiter queue []
Shared Connection information (shared partitions 200)
com.ibm.ws.tx.jta.TransactionImpl@6cb16cb1#tid=6 **MCWrapper id 58a458a4** Managed connection
WSRdbManagedConnectionImpl@69296929 State:STATE_TRAN_WRAPPER_INUSE **Thread Id: 0000001d** Thread Name:
WebContainer : 5 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@6cb16cb1#tid=6
com.ibm.ws.tx.jta.TransactionImpl@2c942c94#tid=5 MCWrapper id 49484948 Managed connection
WSRdbManagedConnectionImpl@15aa15aa State:STATE_TRAN_WRAPPER_INUSE Thread Id: 0000001c Thread Name:
WebContainer : 4 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@2c942c94#tid=5
com.ibm.ws.tx.jta.TransactionImpl@232c232c#tid=1 MCWrapper id 420042 Managed connection
WSRdbManagedConnectionImpl@32ba32ba State:STATE_TRAN_WRAPPER_INUSE Thread Id: 00000018 Thread Name:
WebContainer : 0 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@232c232c#tid=1
com.ibm.ws.tx.jta.TransactionImpl@76d676d6#tid=3 MCWrapper id 64a064a Managed connection
WSRdbManagedConnectionImpl@19441944 State:STATE_TRAN_WRAPPER_INUSE Thread Id: 0000001a Thread Name:
WebContainer : 2 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@76d676d6#tid=3
com.ibm.ws.tx.jta.TransactionImpl@775b775b#tid=7 MCWrapper id 58885888 Managed connection
WSRdbManagedConnectionImpl@523f523f State:STATE_TRAN_WRAPPER_INUSE Thread Id: 0000001e Thread Name:
WebContainer : 6 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@775b775b#tid=7
Total number of connection in shared pool: 5
Free Connection information (free distribution table/partitions 5/1)

No free connections

UnShared Connection information
No unshared connections

Root Cause #4 – Long running queries

- Trace data shows global transactions with shareable connections or LTCs with unshareable connections (not Root Cause #1)
- Each connection allocated to different thread (not Root Cause #2)
- Connections eventually close (not Root Cause #3)

```
[9/3/13 21:33:40:431 EDT] 00000018 PoolManager   3   release(), Pool contents ==>
                          PoolManager name:jdbc/db2demo
PoolManager object:1581276736
Total number of connections: 5 (max/min 5/0, reap/unused/aged 180/1800/0, connectiontimeout/purge 180/EntirePool)
                          (testConnection/inteval false/0, stuck timer/time/threshold 0/0/0, surge time/connections 0/-1)
Shared Connection information (shared partitions 200)
  No shared connections

Free Connection information (free distribution table/partitions 5/1)
  (0)(0)MCWrapper id 58a458a4  Managed connection WSRdbManagedConnectionImpl@69296929  State:STATE_ACTIVE_FREE
Used with transaction com.ibm.ws.tx.jta.TransactionImpl@6cb16cb1#tid=6
  (0)(0)MCWrapper id 49484948  Managed connection WSRdbManagedConnectionImpl@15aa15aa  State:STATE_ACTIVE_FREE
Used with transaction com.ibm.ws.tx.jta.TransactionImpl@2c942c94#tid=5
  (0)(0)MCWrapper id 420042  Managed connection WSRdbManagedConnectionImpl@32ba32ba  State:STATE_ACTIVE_FREE Used
with transaction com.ibm.ws.tx.jta.TransactionImpl@232c232c#tid=1
  (0)(0)MCWrapper id 64a064a  Managed connection WSRdbManagedConnectionImpl@19441944  State:STATE_ACTIVE_FREE Used
with transaction com.ibm.ws.tx.jta.TransactionImpl@76d676d6#tid=3
  (0)(0)MCWrapper id 58885888  Managed connection WSRdbManagedConnectionImpl@523f523f  State:STATE_ACTIVE_FREE Used
with transaction com.ibm.ws.tx.jta.TransactionImpl@775b775b#tid=7

  Total number of connection in free pool: 5
```

Root Cause #4 – Long running queries

- Check activity on threads that held connections, for example thread 0000001d
- Last activity on thread is:

  [9/3/13 21:29:40:693 EDT] 0000001d WSJdbcPrepare > executeQuery Entry
  com.ibm.ws.rsadapter.jdbc.WSJdbcPreparedStatement@42084208
- executeQuery call returns 4 minutes later:

  [9/3/13 21:33:40:364 EDT] 0000001d WSJdbcPrepare < executeQuery Exit
  com.ibm.ws.rsadapter.jdbc.WSJdbcResultSet@56245624
- Look for prepareStatement call to see the query:

  [9/3/13 21:29:40:570 EDT] 0000001d WSJdbcConnect > prepareStatement Entry
  com.ibm.ws.rsadapter.jdbc.WSJdbcConnection@104b104b
  select slow_query(240) from dual
  TYPE FORWARD ONLY (1003)
  CONCUR READ ONLY (1007)
- Check other threads to find other long-running executeQuery() or executeUpdate() calls, or other long-running operations
- Use IBM Trace and Request Analyzer tool
  - Available from developerWorks
  - Can quickly find long-running method calls, including JDBC queries and updates

How to resolve Root Cause #4

- Work with DBA and/or database vendor support team to determine why statement is taking longer than expected to execute
  - JDBC driver tracing might need to be enabled, can be enabled within WebSphere Application Server
- Try executing same statement from a JDBC program outside of application server on the same machine, using same JDBC driver and properties specified in data source
- Consider setting webSphereDefaultQueryTimeout property
- If long-running statements are expected, maximum connection pool size may need to be increased

Root Cause #5 – Max Connections too low

- Maximum Connections set too low

```
[9/3/13 22:20:51:322 EDT] 00000027 PoolManager   3   reserve() entry, Pool contents ==> PoolManager name:jdbc/db2demo
PoolManager object:716843706
Total number of connections: 5 (max/min 5/0, reap/unused/aged 180/1800/0, connectiontimeout/purge 180/EntirePool)
                    (testConnection/inteval false/0, stuck timer/time/threshold 0/0/0, surge time/connections 0/-1)
The waiter count is 45
The mcWrappers in waiter queue []
Shared Connection information (shared partitions 200)
   com.ibm.ws.tx.jta.TransactionImpl@57b657b6#tid=2  MCWrapper id 507c507c  Managed connection
WSRdbManagedConnectionImpl@59895989  State:STATE_TRAN_WRAPPER_INUSE Thread Id: 0000001a Thread Name:
WebContainer : 1 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@57b657b6#tid=2
   com.ibm.ws.tx.jta.TransactionImpl@452d452d#tid=8  MCWrapper id 71b671b6  Managed connection
WSRdbManagedConnectionImpl@109b109b  State:STATE_TRAN_WRAPPER_INUSE Thread Id: 00000020 Thread Name:
WebContainer : 7 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@452d452d#tid=8
   com.ibm.ws.tx.jta.TransactionImpl@6dab6dab#tid=9  MCWrapper id 68ab68ab  Managed connection
WSRdbManagedConnectionImpl@1cc31cc3  State:STATE_TRAN_WRAPPER_INUSE Thread Id: 00000021 Thread Name:
WebContainer : 8 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@6dab6dab#tid=9

   com.ibm.ws.tx.jta.TransactionImpl@50a150a1#tid=4  MCWrapper id 1d201d2  Managed connection
WSRdbManagedConnectionImpl@5fb35fb3  State:STATE_TRAN_WRAPPER_INUSE Thread Id: 0000001c Thread Name:
WebContainer : 3 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@50a150a1#tid=
   com.ibm.ws.tx.jta.TransactionImpl@1e811e81#tid=7  MCWrapper id 2dc82dc8  Managed connection
WSRdbManagedConnectionImpl@58745874  State:STATE_TRAN_WRAPPER_INUSE Thread Id: 0000001f Thread Name:
WebContainer : 6 Handle count 1 Used with transaction com.ibm.ws.tx.jta.TransactionImpl@1e811e81#tid=7
  Total number of connection in shared pool: 5
Free Connection information (free distribution table/partitions 5/1)

  No free connections

UnShared Connection information
  No unshared connections
```

Root Cause #5 – Max Connections too low

- Trace data shows global transactions with shareable connections or LTCs with unshareable connections (not Root Cause #1)
- Each connection allocated to different thread (not Root Cause #2)
- Connections eventually close (not Root Cause #3)
- No long running JDBC statements or other operations (not Root Cause #4)
- There might be more load than can be handled with the current Maximum connection pool size
- Check values of Web Container thread pool size (default = 50) and ORB thread pool size for EJB requests (default = 50)
- Ensure that “Allow thread allocation beyond maximum thread size ” checkbox is not selected for the thread pools
- Is likely the root cause if problem only occurs under high load and goes away when load is reduced

How to resolve Root Cause #5

- Increase Maximum Connections to a value equal to or greater than the maximum Web Container and/or ORB thread pool sizes
- Run thorough load testing to find ideal value for Maximum Connections
- Run trace or showPoolContents to find the highest “waiter count” and add that to current value of Maximum Connections
- Enable PMI and use connection pool counters in Tivoli Performance Viewer within the administrative console to find best value
  - Counters include PoolSize, FreePoolSize, WaitTime, WaitingThreadCount, AllocateCount, ReturnCount

General Suggestions

- If the problem cannot be reproduced and data cannot be collected, follow these general guidelines to reduce likelihood of Connection Wait Timeouts
  - Test with higher values for Maximum Connections
  - Ensure that Maximum Connections is equal to or greater than maximum web container and/or ORB thread pool sizes
  - Check application for connection leaks, especially cases where there is no finally block to close connections
  - Ensure that application follows get – use – close pattern
  - Switch from shareable to unshareable connections if application does not use global transactions

Summary

- Overview of Connection Wait Timeouts
- Problem Symptoms
- Data Collection
- Data Analysis
  - Five common root causes
    - Use of Shareable Connections in Long-Running LTC
    - Multiple connection requests on same thread without calling close()
    - Connection leak in application
    - Long-running queries or slow database response time
    - Maximum Connections set too low
  - How to resolve problem for each root cause
- Questions and Answers

Additional Resources

- Technical Exchange: "Understanding J2C Connection Pooling in WebSphere Application Server"
  http://www.ibm.com/support/docview.wss?uid=swg27039167
- Technical Exchange: "Response Time Analysis for Databases and Web Services in WebSphere Application Server"
  http://www.ibm.com/support/docview.wss?uid=swg27016472
- MustGather: Connection pooling problems for WebSphere Application Server
  http://www.ibm.com/support/docview.wss?uid=swg21254645
- Technote: "How to troubleshoot J2CA0045E connection pooling problems"
  http://www.ibm.com/support/docview.wss?uid=swg21385033
- Education Assistant: "Connection pool tuning and management problems"
  http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.was_v7/was/7.0/ProblemDetermination/WA5721G10_ConnectionPool_edited/player.html

Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at:
  http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
  http://www.ibm.com/developerworks/websphere/community/
- Join the Global WebSphere Community:
  http://www.websphereusergroup.org
- Access key product show-me demos and tutorials by visiting IBM® Education Assistant:
  http://www.ibm.com/software/info/education/assistant
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically:
  http://www.ibm.com/software/websphere/support/d2w.html
- Sign up to receive weekly technical My Notifications emails:
  http://www.ibm.com/software/support/einfo.html

Connect with us!

1. **Get notified on upcoming webcasts**
   Send an e-mail to wsehelp@us.ibm.com with subject line “wste subscribe” to get a list of mailing lists and to subscribe
2. **Tell us what you want to learn**
   Send us suggestions for future topics or improvements about our webcasts to wsehelp@us.ibm.com
3. **Be connected!**
   Connect with us on Facebook
   Connect with us on Twitter

Questions and Answers