The Bug

22572201 Matthew Tam

September 12, 2022

1 Description of the bug

The bug occurs when one thread is creating a `DatabaseConnectionPool` and another thread is calling the database connection pool's `acquire` function, shown in the listing 1.

The problem is that when `thread 0` calls `aquireConnection` 2, aquireConnection retrieves the `wrapper lock`, then waits to acquire the `_connectionPoolLock`. While `thread 0` is calling the `aquireConnection` function, `thread 1` is constructing a `DatabaseConnectionPool`. As a database connection pool is created a `DatabaseConnectionPoolMonitor` is created, a thread is then made for this monitor, and the run function of this thread calls a `refresh` function. The refresh function then calls a `distributeConnection` function 3 and this function starts the deadlock. The distributeConnection function acquires the `_connectionPoolLock` and then calls the wrapper function `setConnection` 4 which then waits to acquire the `wrapper lock`. Thus, a deadlock has occurred because `thread 0` is waiting for a lock that `thread 1` has but `thread 1` is waiting for a different lock that `thread 0` is holding.

2 Recreating the bug

Assuming that the Connection Pool file is available and the JPF git repository is cloned. Adding the Connection Pool Java file to `jpf-core\src\examples\`, then creating a new Java file and creating a public class and a main function within that file. Then creating a `DatabaseConnectionPool` and then creating a `DatabaseConnection` by using the `acquireConnection` function on the pool created. The recreation is shown in 1. The next step is to create a `.jpf` file to give jpf properties to find the deadlock. The properties to look at are; file name in which to look for the bug, breath first search instead of depth first search, choosing a random choice seed, and a search limit. The listed properties are show in the list 5.

```
public class DBDriverMatthewTam {
    /**
     * @param args
     * @throws DatabaseException
     */
    public static void main (String[] args) throws DatabaseException {
            DatabaseConnectionPool Pool1 = new DatabaseConnectionPool("pool1",
            "jdbc:oracle:thin:@localhost:1521:orcl", "scott", "tiger", 10, 5);
            DatabaseConnection dc1 = Pool1.acquireConnection(40);
    }
}
```

Listing 1: Driver

```
    DatabaseConnection acquireConnection( long timeout ) {
        if (_poolMaxSize <= 0) {
            try {
                return new DatabaseConnection(null, _url, _user, _password, _name);
            } catch (Exception e) {
                return null;
            }
        }

        DatabaseConnection result = null;

        DatabaseConnectionWrapper wrapper = new DatabaseConnectionWrapper();

        synchronized (wrapper)
        {
          boolean needWait = true;
          try{
              long t1 = 0, t2 = 0;
              synchronized (_connectionPoolLock)
              {
                  if ((_connections.size() > 0) && (_waiting.size() == 0))
                  {
                      needWait = false;
                      result = (DatabaseConnection) _connections.remove(0);
                  }
                  else
                  {
                      _waiting.add(wrapper);
                  }
              }

              if (needWait) {
                  result = wrapper.waitForConnection(timeout);
              }
            }finally{
              if (needWait) {
                  synchronized (_connectionPoolLock) {
                      _waiting.remove(wrapper);
                  }
              }
            }
        }

        return result;
    }
```

Listing 2: aquireConnection

```
void distributeConnection(DatabaseConnection c)
    {
        DatabaseConnectionWrapper wrapper = null;

        long t1 = 0, t2 = 0;
        synchronized (_connectionPoolLock)
        {
            while( _waiting.size() > 0 ) {
                wrapper = (DatabaseConnectionWrapper) _waiting.remove(0);
                if( wrapper.setConnection( c ) == true ) {
                    break;
                }
                wrapper = null;
            }
            if( wrapper == null ) {
                _connections.add(0,c);
            }
        }
        if (wrapper != null) {
            synchronized( wrapper ) {
                wrapper.notifyAll();
            }
        }
    }
```

Listing 3: distributeConnection

```
        public boolean setConnection( DatabaseConnection connection ) {
            synchronized( this ) {
                if( _isWaiting && _connection == null ) {
                    // wrapper is waiting and does not have a connection yet
                    _connection = connection;
                    return true;
                }
                else {
                    return false;
                }
            }
        }
```

Listing 4: setConnection

```
target = DBDriverMatthewTam
search.class = .search.heuristic.BFSHeuristic
cg.randomize_choices = VAR_SEED
search.depth_limit = 50
```

Listing 5: .jpf file