

Re: ODBC/OLE DB Connection Pool

Source: <http://www.tech-archive.net/Archive/Data/microsoft.public.data.oledb/2004-11/0147.html>

From: Bob Barrows [MVP] (*reb01501_at_NOyahoo.SPAMcom*)

Date: 11/26/04

Date: Fri, 26 Nov 2004 13:20:04 -0500

Thank you. I now see where you got that idea.

Kevin Joseph wrote:

- > *Hi Bob,*
- >
- > *Reason I think the error is related to connection pooling is cos of*
- > *the article on MS support site*
- > <http://support.microsoft.com/default.aspx?scid=kb:en-us:328476>
- >
- > *The web site I have servers 20,000+ users. Out of 100,000 hits to the*
- > *IIS server around 200 or at most 2000 fail with the "server not found"*
- > *error message. The above article clearly states that it is related to*
- > *Connection Pool (I have already completed the TCP/IP tasks without*
- > *success)*

No, that's not what the article says. It says that one possible outcome from turning off pooling is the errors you are seeing.

"Note that you can also receive these specific error messages when other problems are occurring with SQL Server; for example, you may receive these error messages if the remote computer that is running SQL Server is shut down, if the remote computer that is running SQL Server is not listening to TCP/IP sockets at all, if network connectivity to the computer that is running SQL Server is broken because the network cable is pulled out, or if you are having DNS resolution issues."

- > *If connection pool is setup on the client, it is possible to monitor*
- > *the SQL server connections using perfmon - if the connections grow ;*
- > *it means that connection pooling is not working.*

I'm not sure that this is the correct conclusion. It may be, but other conclusions are possible. For example, are you using a single login id for all your applications that communicate with the database server? If not, you will get connections for each login id that your applications use. Also, since your server experiences heavy usage, pooled connections may not be available when needed, resulting in the creation of more connections.

- > *And this is exactly*
- > *what is happening in my case. If you have some other way of proving*
- > *that connection pooling is fine ; I will gladly accept your point of*
- > *view.*

In IIS 4 and higher, using MDAC2.1 and higher, pooling is on unless you've turned it off.

I think this section from the article I keep citing is relevant

(<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmdac/html/pooling2.asp>):

Maintaining a Persistent Connection for Resource Pooling

This article has repeatedly emphasized the need for developers who use OLE DB resource pooling to keep at least one connection open for each set of user credentials used to access the data store. This is not to be confused with the mistake of opening up multiple and unnecessary Connection objects, or even multiple Recordset objects. It's also not to be confused with using just one Connection object for several hundred Active Server Pages (ASP pages).

ASP developers should open one connection per set of unique user credentials. However, an ASP developer can eliminate the benefits of pooling in this scenario in the following ways:

Instantiating dozens or even hundreds of instances of ADO objects at once.

For example, you can easily circumvent any performance gains found in pooling by using code such as the following:

```
Dim cnn(200) As ADODB.Connection  
Dim rst(200) As ADODB.Recordset
```

[my comment]: this is VB code, not vbscript. There is no way this code could be used in an ASP page. It could, however, be used in a VB COM object being called by vbscript code in and ASP page. Also, it could be written as:

```
dim cnn(200)  
dim rst(200)
```

which is perfectly acceptable vbscript code, and the point of this section would be relevant.

Presently, these two lines of code appear to be frequently and incorrectly used on individual ASP pages. The overhead of instantiating, opening, and manipulating this many objects will eliminate any benefit of pooling, not to mention that it will swell the pool to unnecessary size just to hold each of those connections. If you use this technique on multiple ASP pages, which in turn are hit by multiple users, the amount of memory needed just to hold all of those ADO objects in memory can soon reach into the gigabytes.

Using just one Connection object for multiple ASP pages.

If you don't want to circumvent the benefits of pooling, however, you can do this: Within a given ASP page, open the Connection object and one or more Recordset objects that you need. Then close and delete them.

This also applies for non-ASP developers who are developing an application with multiple threads. Do not try to share a single Connection object between all threads. Instead, use one thread, one Connection object, and one persistent connection in your main application to keep the pool alive.

For Microsoft Transaction Server developers, this is not an issue. Microsoft Transaction Server itself enables the pool, whether you implement a single Connection object or not. Because Microsoft Transaction Server is inherently stateless, trying to keep that persistent Connection object around, let alone actually use it, is redundant.

If you use Server.CreateObject, you will be using MTS.

And this section:

Top 10 Reasons Why OLE DB Resource Pooling Might Not Work

To summarize what this article has discussed, here is a list of the top 10 reasons why pooling might not be turned on. This text was first made available in the OLE DB Readme file.

The registry value OLEDB_Services must be present under the provider's HKEY_CLASSES_ROOT/<CLSID> key. OLEDB_Services is most commonly set to 0xffffffff, or to desired bits of DBPROPVAL_OS_*. If this does not exist or is set to 0x00000000, pooling will never occur. For more information, refer to "Setting Provider Service Defaults" in the OLE DB Services documentation.

[my comment] None of the following can be done in ASP, except by calling a COM object in which one of these actions is done

The consumer can override the OLEDB_Services key and disable pooling by setting DBPROP_INIT_OLEDBSERVICES.

For more information, refer to "Overriding Provider Service Defaults" in the OLE DB Services documentation.

The provider must be free-threaded. If a provider developer uses the OLE DB provider templates for Visual C++ or Visual Basic, the templates will not create free-threaded providers by default. In those cases, pooling will be disabled regardless of what the property or registry indicates.

Setting DBPROP_INIT_PROMPT and DBPROP_INIT_HWND disables pooling of a data source object. These properties must be set to either VT_EMPTY or NOPROMPT.

Consumers cannot use QueryInterface for any interface unknown to service components prior to initialization. That is, applications that need to pool should not use QueryInterface, as COM suggests, to see what interfaces the provider supports. For example, a simple QI for IDBAsynchStatus, to determine whether the provider is Asynch or not, eliminates the data source

object from the pool.

Calling IDBProperties::GetPropertyInfo prior to initialization disables pooling.

If IDBInitialize::UnInitialize is called, the released data source object will not be pooled.

Pooling does not occur on Windows 95.

Providers must correctly implement aggregation.

Do not use data source object notifications. If you use a QueryInterface for IConnectionPointContainer to advise a listener to the data source object, the object will never be pooled.

NOTE These items are cumulative. That is, any item will disable pooling for that data source object, regardless of what the other values are.

Bottom line: I think your pooling is turned on, unless some serious problem exists on your server, or applications are running on your server which are turning it off.

Bob Barrows

--

Microsoft MVP - ASP/ASP.NET

Please reply to the newsgroup. This email account is my spam trap so I don't check it very often. If you must reply off-line, then remove the "NO SPAM"