2 Tools to Keep SQL Server Tuned

OSTRESS and Read80Trace help you ensure that queries perform their best

Hari Ramachandran | *SQL Server Pro* *Sep 19, 2005*

Downloads
47465.zip

For years, Microsoft Product Support Services (PSS) has used two tools—OSTRESS and Read80Trace—to simulate scenarios and analyze large SQL Server trace files for its customers. At the Professional Association for SQL Server (PASS) 2004 conference, PSS released these tools to the public. SQL Server DBAs and developers will find OSTRESS useful when working on complex stress-testing scenarios and Read80Trace helpful for analyzing SQL Server trace files to troubleshoot performance issues. I'll provide some detailed usage scenarios for these tools and give you pointers for using them effectively.

Getting Started

First download the OSTRESS and Read80Trace utilities from the download link in the Microsoft article "Description of the SQL Server Performance Analysis Utilities Read80Trace and OSTRESS" at http://www.support.microsoft.com/?kbid =887057. By default, Windows installs the tools (ostress.exe and read80trace.exe) and the Help files (ostress.chm and read80trace.chm) in the C:\rml folder. Both tools' Help files provide useful documentation and design overviews. You can find additional information about using the tools at Microsoft's PSS Service Center Labs Web page at http://www.microsoft.com/downloads/details.aspx?familyid=aec18337-887f-4ec6-a858-81f84de8082f&displaylang=en.

To use OSTRESS to stress-test SQL Server, you need to specify a query or script file to run several times over multiple simultaneous connections. For example, to simulate a scenario of five simultaneous connections running the same query on the pubs database, you'd use the following syntax to connect to a local server and execute a query:

```
ostress -SServerName -E -dpubs                    -Q"SELECT * FROM authors" -n5
```

(Commands wrap to multiple lines here because of space constraints; you should type the command on one line.) The parameters for OSTRESS are similar to those for the OSQL utility and are case-sensitive. You use the -S parameter to specify the server to connect to, -E to specify Windows authentication, -d to specify the database in which the query needs to be run, and -Q to specify the query to run. The -n5 parameter (-n specifies the number of threads that will be spawned concurrently to run each input file or query) tells OSTRESS to open five simultaneous connections to SQL Server and execute the query in each of them. The OSTRESS Help file contains a complete list and description of the command parameters.

Alternatively, you can specify a batch file as input to OSTRESS by using the -i parameter, like this:

```
ostress -SServerName -E -dpubs                    -ic:\temp\test\batch1.sql
```

Here the batch file (batch1.sql) contains the statement

```
SELECT * FROM Authors
```

The -r5 parameter specifies that each connection will run the batch for five iterations. Therefore, this OSTRESS command would run the query 25 times (5 threads * 5 iterations). By using the -n and the -r parameters, you can immediately see the OSTRESS utility's usefulness in simulating a load on a SQL Server system.

BLOG APR 21, 2016

Turning Global Addresses into your Biggest Asset

Sponsored Blog Are you dealing with national and international records? The more expansive your records are in territory, the more problems you will face when it comes to verifying addresses. Not only are you dealing with language differences, but you are also dealing with different address formatting for different countries....More

BLOG MAR 14, 2016

Building Your Business with Quality Data

Your contact data is valuable, but is it up to its full potential? Without the proper maintenance, the quality of your data quickly decreases--so quickly that 50% of databases deteriorate after only two years. It is an absolute necessity to maintain your data in order to

Using OSTRESS to Simulate Random Timeouts

Another way you can use OSTRESS is to simulate query timeouts, which you do by specifying the -t parameter. This technique is useful in testing a scenario in which a user cancels a query or the query times out and leaves orphaned transactions in SQL Server—for example, when the application doesn't implement proper error handling for timeouts or query cancellation and transactions that weren't rolled back cause blocking and other concurrency problems. This sample command runs the batch file over 10 concurrent connections, five iterations per connection, with a 1-second query timeout:

```
ostress -SServerName -E -dpubs                                    -ic:\temp\test\batch1.sql
                                                                  -n10 -r5 -t1
```

When you run this command, you'll notice that some timeouts in the output look similar to those in the output in Figure 1 shows.

You can also configure OSTRESS to randomly simulate timeouts or simulate timeouts a certain percentage of the time. You can find more information about these techniques under the Random Events topic in the OSTRESS Help file. To simulate timeouts a certain percentage of the time, use the CancelPct configuration value in the control file and pass the control file as the -c parameter to the OSTRESS command. The \rml directory contains a sample control file (sample.ini) that provides examples of setting advanced options. The following sample command demonstrates using the control file to set the behavior of the OSTRESS execution:

```
ostress -SServerName                              -E -dpubs                              -ic:\temp\test\batch1.sql                              -csample.ini
```

The settings in a sample.ini file to cancel queries 10 percent of the time would look like this:

```
\[Query Options\]                              CancelPct=10.00
```

You can find more documentation of the different configuration options under the Control File topic in the Help file.

Using OSTRESS to Reproduce Deadlock Scenarios

Because deadlock scenarios tend to be highly time-sensitive, trying to reproduce them on development or test machines with few users and scant data can be tricky. If you suspect which queries and procedures are deadlocking, you can create a batch file that includes those queries and procedures and try to simulate the deadlocking by running the batches through multiple connections and iterations. Figure 2 shows a small batch file that you can run from a command prompt to simulate a simple deadlocking scenario on the sample pubs database.

The sample scripts in Listings 1 and 2 deliberately simulate deadlocking. Listing 1 updates the Authors table, then the Titles table; Listing 2 performs these actions in reverse. When you run deadlock.bat, which starts two instances of OSTRESS, you'll notice that some of the connections generate error messages like the one that Figure 3 shows.

To troubleshoot the deadlocking issue quickly, you can run a SQL Server Profiler trace (or use the sp_trace_* procedure to run a server-side trace) at the same time you run the OSTRESS command or turn on the deadlocking trace flags by running the command

```
DBCC TRACEON(1204,3605,-1)
```

These deadlock trace flags will generate detailed deadlock information in the SQL Server error log file. You can disable the trace flags by running the DBCC TRACEOFF command.

Using OSTRESS to Reproduce RPC Events

Most applications submit queries to SQL Server as remote procedure call (RPC) events instead of SQL:Batch events. If you want to reproduce a similar behavior from within OSTRESS or OSQL, you can use the ODBC call syntax to do so (especially if you can reproduce a specific problem only through the affected application and not from Query Analyzer).

To capture queries as RPC events, you need to include the ODBC call escape-clause syntax in the batch file you provide to OSTRESS, as the following example shows:

```
\{call pubs.dbo.byroyalty (10)\}
```

When you run this statement in a tool such as Query Analyzer and view a Profiler trace of the query, you'll see that the query is recorded in the trace as an RPC event, as Figure 4 shows. The following query uses the regular EXEC syntax and is recorded in the trace as a SQL:Batch event, as Figure 5 shows.

```
EXEC pubs.dbo.byroyalty 10
```

For more information about using ODBC calls, see the SQL Server *Books Online—>BOL—* topic *How to call stored procedures (ODBC)*.

Read80Trace Simplifies Trace Analysis

As a DBA, you probably often find yourself striving to improve the performance of SQL Server queries. Typically, you do so by analyzing large SQL Server trace files, trying to discover the queries or batches that take too long to run, perform too many I/O requests, or use too many CPU cycles. Although you can open SQL Server traces in Profiler and sort them by different columns (e.g., duration, cpu, reads, writes), this process tends to be time-consuming and doesn't provide aggregated data. Alternatively, you could load the traces into SQL Server tables and run aggregate queries against these tables to analyze the worst-performing queries in the trace. The Read80Trace utility simplifies trace analysis by automatically creating an analysis database for the trace file(s) you provide. It also generates a graphical HTML output file that contains detailed information about the load captured in the trace files. Let's examine how to capture a SQL Server trace and use Read80Trace to analyze its contents.

Capturing a SQL Server Trace

Before you use Read80Trace, you must first capture a SQL Server trace to run the utility against. When you capture a Profiler trace for use with Read80Trace, you need to capture several events and columns. If you use the SQLProfilerStandard default Profiler template (SQLProfilerStandard.tdf) to capture trace information, make sure that you add the EndTime and DatabaseId columns to the template before capturing the trace. If you want statement-level analysis, you need to select statement-level events such as SP:StmtStarting, SP:StmtCompleted, and Showplan Statistics when you create the trace. You can find more information about the events and columns you need to capture in a trace in the Help file (read80trace.chm) under the topic Necessary Events And Columns For Performance Analysis.

I prefer to use the trace stored procedures (e.g., sp_trace_create, sp_trace_setevent) to capture the SQL trace instead of using the Profiler trace GUI utility because the stored procedures cause less overhead on the server than Profiler. An easy way to generate an SQL script that uses the trace procedures to run a trace is to use the Script Trace option in Profiler and run the generated script on your SQL Server system. Another option for collecting Profiler trace information is to use the PSSDiag diagnostic data-collection utility, which is documented in the Microsoft article "PSSDIAG data collection utility" at http://support.microsoft.com/?kbid=830232.

After you've obtained a SQL Server trace file that contains the collected events, run Read80Trace against the trace file by executing a command similar to this:

```
read80trace                                -ic:\temp\test\sample_trace.trc                                -ic:\temp\test\sample_trace.trc
```

The Read80Trace utility parses the specified trace file, creates an output.htm file in the specified output folder, and opens the file in your browser. The utility connects to a SQL Server system (by default, to the local server if no parameter is specified, or to a server name you specify with -S) and creates a database called PerfAnalysis (the default) or a nondefault database name, which you specify with the -d option. Read80Trace connects to the SQL Server system to perform extended analysis and aggregations on the trace file and generates a read80trace.log file. This log file records useful troubleshooting information about the utility, such as the server it connected to, the number of events it processed, and warnings. Because the trace analysis can be CPU-intensive, you should use a test (i.e., nonproduction) SQL Server system for Read80Trace to connect to.

Read80Trace Output

The output.htm file that Read80Trace generates contains helpful analysis information, such as rollups by batch duration, CPU, reads, and writes. If the trace captured statement-level events, the output file contains rollups by statement duration, CPU, reads, and writes. You can use this output to quickly identify slow-running and resource-intensive queries in the system.

The Read80Trace utility internally produces Replay Markup Language (RML) files named SQL*nnnnn*.rml (where *nnnnn* is a number that represents the server process ID—SPID—for each process) in the output directory. The RML files are XML files that contain detailed information about each SPID, such as connection and query information. You can use RML files to control how and when commands are submitted to a SQL Server system from a text document; RML also serves as a bridge between Read80Trace, OSTRESS, and related utilities. You can find more information about RML's design and the utilities associated with it in the utilities' Help files.

When you run the Read80Trace command, you can specify any of three options for displaying trace output:

- -M—This option breaks out the trace file by SPIDs, so you'll see multiple files (e.g., SPID00051.trc, SPID00052.trc) in the output directory. This option is useful when you need to analyze a problem from a specific trace and you know the SPID of the process involved.
- -r##—If you have a large number of trace files that you generated sequentially by using the TRACE_FILE_ROLLOVER option in the sp_trace_create procedure, they'll be named filename.trc, filename_1.trc, filename_2.trc, and so on. In this case, you can use the -r## option to start analyzing the traces sequentially. The number symbols (##) represent the number of trace files in the series following the first trace file you specified.
- -d *databasename*—You can use this option to specify a database name (instead of the default PerfAnalysis database) to store aggregate information. You might find this option useful when you're analyzing multiple trace files and want to store their information in separate databases.

The Read80Trace Help file provides more information about tables, views, and queries used to generate the trace analysis as well as a complete list of Read80Trace parameters. As I mentioned earlier, the Help file provides a design overview of the utility and also includes an entity relationship (ER) diagram of the tables in the PerfAnalysis database.

Advanced PerfAnalysis Queries

If you want a more detailed analysis of the trace information than what the output.htm file provides, you can run queries against the PerfAnalysis database and analyze the results of those queries. The PerfAnalysis database has multiple views that provide information about the queries in the trace file. Note that the generatexml.sql file in the output directory shows what queries are executed against this database to generate the aggregate information that's in the default output.htm file. For example, the query in Listing 3 joins two views in the database to generate the "Rollup_Batch_Duration" output.

You could write your own PerfAnalysis database queries and generate custom analysis information. For example, the query in Listing 4 generates a list of batches that received an attention signal or a query-cancellation notification during the time the trace was captured.

You can use the Read80Trace and OSTRESS tools in several other advanced stress-testing and analysis scenarios. For example, you can use the .rml files that the Read80Trace file produces as input to the OSTRESS utility to replay the commands in those files, as this sample command shows:

```
ostress -E -dpubs                    -ic:\temp\test\output                    \SQL00139.rml                    -SServerName -n
```

In this example, the .rml file that Read80Trace generated for SPID 139 is input to the OSTRESS utility, and the queries that SPID 139 generated are run over five concurrent connections.

Useful Analysis Tools

I've explained how you can use the OSTRESS tool to generate specific stress loads on your SQL Server system, which can reveal performance trouble spots such as slow-running queries, blocking, and deadlocking issues that you can detect and analyze before putting your application into production. I've also shown you how to use the Read80Trace tool to analyze trace statistics to tune database and query performance. You'll find these highly useful tools valuable additions to your DBA toolkit.

Related Articles

Keeping Concurrent

Reusing Query Plans 1

Leverage Clustered Indexes to Avoid Bookmark Lookups 4

SQL Server 2005 Management Tools 1

Database Design for Performance 3