SQL Server Performance Statistics Using a Server Side Trace

Written By: Greg Robidoux -- 11/26/2012

Problem

When troubleshooting a SQL Server performance problem, one of the tools to use is Profiler. This tool allows you to collect metrics on statements that are run on your SQL Server for analysis and troubleshooting. The problem with using Profiler is that it is a client tool and unless it is run on the server itself the connection may be lost and your trace stops. This usually happens right before the problem your trying to troubleshoot occurs and you don't end up collecting that valuable information you need.

Solution

One alternative to using Profiler is to run a Server Side Trace. This process runs on the server and collects trace statistics pretty much the same way that you do using Profiler, but the process involves running a T-SQL script to start, run and stop the trace vs. using the Profiler GUI.

The server side trace can be modified to collect any event that the trace process can capture, but for this example we are just looking at SQL:StmtCompleted events which occur when a T-SQL statement has completed. For a complete list of events click here.

EventNumber	Event	Description
41	SQL:StmtCompleted	Occurs when the Transact-SQL statement has completed.

In addition to collecting information on certain events, you can also specify what data to collect. In this example we are collecting the statements or TextData, the SPID, Duration etc... For a complete list of columns click here.

ColumnNumber	Column	Description
1	TextData	Text value dependent on the event class that is captured in the trace.
12	SPID	Server Process ID assigned by SQL Server to the process associated with the client.
13	Duration	Amount of elapsed time (in milliseconds) taken by the event. This data column is not populated by the Hash Warning event.
14	StartTime	Time at which the event started, when available.
15	EndTime	Time at which the event ended. This column is not populated for starting event classes, such as **SQL:BatchStarting** or **SP:Starting**. It is also not populated by the **Hash Warning** event.
16	Reads	Number of logical disk reads performed by the server on behalf of the event. This column is not populated by the **Lock:Released** event.
17	Writes	Number of physical disk writes performed by the server on behalf of the event.

To create the trace for these events and columns the command would look as follows:

```
/****************************************/
/* Server Side Trace                    */
/****************************************/
-- Declare variables
DECLARE @rc INT
DECLARE @TraceID INT
DECLARE @maxFileSize bigint
```

```
DECLARE @fileName NVARCHAR(128)
DECLARE @on bit

-- Set values
SET @maxFileSize = 5
SET @fileName = N'C:\TestTrace'
SET @on = 1

-- Create trace
EXEC @rc = sp_trace_create @TraceID output, 0, @fileName, @maxFileSize, NULL

-- If error end process
IF (@rc != 0) GOTO error

-- Set the events and data to collect
EXEC sp_trace_setevent @TraceID, 41,  1, @on
EXEC sp_trace_setevent @TraceID, 41, 12, @on
EXEC sp_trace_setevent @TraceID, 41, 13, @on
EXEC sp_trace_setevent @TraceID, 41, 14, @on
EXEC sp_trace_setevent @TraceID, 41, 15, @on
EXEC sp_trace_setevent @TraceID, 41, 16, @on
EXEC sp_trace_setevent @TraceID, 41, 17, @on

-- Set Filters
-- filter1 include databaseId = 6
EXEC sp_trace_setfilter @TraceID, 3, 1, 0, 6
-- filter2 exclude application SQL Profiler
EXEC sp_trace_setfilter @TraceID, 10, 0, 7, N'SQL Profiler'

-- Start the trace
EXEC sp_trace_setstatus @TraceID, 1

-- display trace id for future references
SELECT TraceID=@TraceID
GOTO finish

-- error trap
error:
SELECT ErrorCode=@rc

-- exit
finish:
GO
```

There are basically four components to this to get this running:

- sp_trace_create - this procedure creates the trace and has 5 parameters
  - TraceID - the ID of the trace
  - Options - various options that can be set
  - TraceFile - physical file name where you want to write the trace file
  - MaxFileSize - size of the file, before closing and creating subsequent files
  - StopTime - time to stop the trace
- sp_trace_setevent - this procedure specifies what event to capture and what column to capture
  - TraceID - the ID of the trace
  - EventID - the ID of the event you want to capture
  - ColumnID - the ID of the column you want to capture
  - On - whether you want to turn this event on or off
- sp_trace_setfilter - this procedure specifies the filters to set. This determines whether you include or exclude data
  - TraceID - the ID of the trace
  - ColumnID - the ID of the column you want to set the filter on
  - LogicalOperator - specifies whether this is an AND or OR operation
  - ComparisonOperator - specify whether the value is equal, greater then, less the, like, etc...
  - Value - the value to use for your comparison
- sp_trace_setstatus
  - TraceID - the ID of the trace
  - Status - stop, start or close a trace

To add additional events and columns you would just include additional sp_trace_setevent commands such as the following to collect event 10 RPC:Completed for the same columns that we were collecting above.

```
EXEC sp_trace_setevent @TraceID, 10,  1, @on
```

```
EXEC sp_trace_setevent @TraceID, 10, 12, @on
EXEC sp_trace_setevent @TraceID, 10, 13, @on
EXEC sp_trace_setevent @TraceID, 10, 14, @on
EXEC sp_trace_setevent @TraceID, 10, 15, @on
EXEC sp_trace_setevent @TraceID, 10, 16, @on
EXEC sp_trace_setevent @TraceID, 10, 17, @on
```

To start, stop and delete a trace you use the following commands.

Task	Command	Notes
To find traceid	SELECT * FROM :: fn_trace_getinfo(default)	This will give you a list of all of the traces that are running on the server.
To start a trace	sp_trace_setstatus traceid, 1	TraceId would be the value of the trace
To stop a trace	sp_trace_setstatus traceid, 0	TraceId would be the value of the trace
To close and delete a trace	sp_trace_setstatus traceid,0 sp_trace_setstatus traceid, 2	To delete you need to stop the trace first and then you can delete the trace. This will close out the trace file that is written.

Once the data has been collected you can load the data into a trace table and then run queries against the trace file. Following are some commands that can be used to load the trace data into a trace table.

Task	Command	Notes

To load a trace	`--Load into a new table`
	`SELECT * INTO sqlTableToLoad`
	`FROM ::fn_trace_gettable('traceFileName', DEFAULT)`
	
	`--Load into an existing table`
	`INSERT INTO sqlTableToLoad`
	`SELECT * FROM ::fn_trace_gettable('traceFileName', DEFAULT)`
		• sqlTableToLoad – replace this with the table where you will load the data to
		• traceFileName – use the correct path for the file that you will be reading the data from. If you are on the server use the UNC path.
		• default – if this is set to default the load will load the file you specified as well as all additional sequenced files that exist. If you want to only load one file change the word 'default' to a number of files you want to load.

To query the table	`SELECT *` `FROM sqlTableToLoad`	

Next Steps

- Using Profiler to trace events that are occurring is a must when troubleshooting performance issues

- Learning how to use Server Side Traces can enhance your performance monitoring process. You can set these up and turn them on and off as needed.
- Add this handy process to your SQL Server toolkit and use all the tools available to maintain your servers.