Analyzing & Optimizing T-SQL Query Performance Part1: using SET and DBCC

Kevin Kline
Senior Product Architect for SQL Server
Quest Software

QUEST SOFTWARE

AGENDA

- Audience Poll
- Presentation

(submit questions to the e-seminar moderator for the Q & A session)

- Product Review

(a few minutes on Quest products for SQL Server)

- Q & A

(answers to questions submitted to the moderator)

QUEST SOFTWARE

ANALYZING & OPTIMIZING QUERY PERFORMANCE WITH SET AND DBCC

This presentation will cover:

- SET STATISTICS IO
- SET STATISTICS TIME
- SET NOCOUNT ON
- DBCC SHOW_STATISTICS
- Index and Table Defragmentation Commands

(DBCC SHOWCONTIG, DBCC INDEXDEFRAG, DBCC DBREINDEX, DBCC SQLPERF)

- DBCC SQLPERF
- DBCC PROCCACHE (with some discussion of DBCC MEMUSAGE and DBCC PINTABLE)

QUEST SOFTWARE

T-SQL TUNING APPROACH

- Recognize the possible sources of slow performance
- Acquire general performance information first, then go after more granular performance information
- Probe for info iteratively
- Capture baseline information as well as troubleshooting information
- Test (and retest) your hypothesis

QUEST SOFTWARE

SET STATISTICS IO

- Enabled before a query is run
- Can be enabled as a query parameter in SQL Query Analyzer
- The important info appears after the result set of the query is returned:
  - How many scans were performed
  - How many logical reads (reads in cache) were performed
  - How many physical reads (reads on disk) were performed
  - How many pages were placed in the cache in anticipation of future reads (read-ahead reads)
- Good queries usually have higher logical reads and few, if any physical reads and scans

QUEST SOFTWARE

EXAMPLE

Query:

```
USE northwind
GO
SET STATISTICS IO ON
GO
SELECT COUNT(*) FROM employees
GO
SET STATISTICS IO OFF
GO
```

Results:

```
-----------
2977

Table 'Employees'. Scan count 1, logical reads
   53, physical reads 0, read-ahead reads 0.
```

QUEST SOFTWARE

TIP!

- Clear the buffer between iterated tests using DBCC DROPCLEANBUFFERS

For example:

```
SET STATISTICS IO ON
GO
SELECT COUNT(*) FROM employees
GO
SET STATISTICS IO OFF
GO
DBCC DROPCLEANBUFFERS
GO
```

QUEST SOFTWARE

SET STATISTICS TIME

- Enabled before a query is run
- Can be enabled as a query parameter in SQL Query Analyzer
- Returns the elapsed time of each query with the query result set
- Depends on the total activity of the server
- Gives you a more accurate metric for the user experience
- Helps you measure software performance in terms of real performance

QUEST SOFTWARE

EXAMPLE #1

Query:

```
SET STATISTICS TIME ON
GO
SELECT COUNT(*) FROM titleauthors
GO
SET STATISTICS TIME OFF
GO
```

Results:

```
SQL Server Execution Times:
   cpu time = 0 ms.  elapsed time = 8672 ms.
SQL Server Parse and Compile Time:
   cpu time = 10 ms.

-----------
25

 SQL Server Execution Times:
   cpu time = 0 ms.  elapsed time = 10 ms.
SQL Server Parse and Compile Time:
   cpu time = 0 ms.
```

QUEST SOFTWARE

EXAMPLE #2

- Use this script to capture time before and after a single command that does not contain multiple GO statements, reporting a total elapsed time in seconds for the statement

Query:

```
DECLARE @start_time DATETIME
SELECT  @start_time = GETDATE()
< any query or a script that you want to
  time, without a GO >
SELECT  'Elapsed sec' = DATEDIFF(second,
  @start_time, GETDATE() )
GO
```

QUEST SOFTWARE

EXAMPLE #3

- Use this script to capture time before and after multiple commands that contain multiple GO statements, reporting a total elapsed time in seconds for the statement

Query:

```
CREATE TABLE #save_time ( start_time DATETIME NOT NULL
   )
INSERT #save_time VALUES ( GETDATE() )
GO

< any script that you want to time (may include GO) >
GO

SELECT 'Elapsed sec' = DATEDIFF(second, start_time,
   GETDATE() )
FROM   #save_time
DROP TABLE #save_time
GO
```

QUEST SOFTWARE

SET NOCOUNT

- Returns the single biggest performance boosts when coding stored procedures, triggers, and functions. Even casual scripting can experience a significant boost!

- Turns of the N rows affected verbiage that appears at the end of every query and eliminates the DONE_IN_PROC internal messaging sent from the server to the client for each step in a stored procedure.

QUEST SOFTWARE

SET NOCOUNT

- Add code to your stored procedures, triggers, and functions to return exactly the strings you wish the user to see rather than rely on default SQL Server behavior

Syntax:

```
CREATE PROC foo
AS
SET NOCOUNT ON

< stored procedure code >

SET NOCOUNT OFF
GO
```

QUEST SOFTWARE

DBCC AND INDEXES

- DBCC offers a variety of functionalities, some corrective and some investigative
- This section shows several DBCC commands that can help investigate the condition and usefulness of table structures and indexes

QUEST SOFTWARE

DBCC SHOW_STATISTICS

DBCC SHOW_STATISTICS offers a very effective way to analyze an indexes effectiveness. The syntax is:

```
DBCC SHOW_STATISTICS ( table_name,
  index_name )
```

Refer to the white paper for an example. (It's pretty long!)

QUEST SOFTWARE

DBCC SHOW_STATISTICS

DBCC SHOW_STATISTICS returns:

- Updated: The date and time the index statistics were last updated
- Rows: The total number of rows in the table
- Rows Sampled: The number of rows sampled for index statistics information
- Steps: The number of distribution steps
- Density: The selectivity of the first index column prefix
- Average key length: The average length of the first index column prefix
- All density: The selectivity of a set of index column prefixes

QUEST SOFTWARE

DBCC SHOW_STATISTICS

- Average length: The average length of a set of index column prefixes
- Columns: The names of index column prefixes for which All density and Average length are displayed
- RANGE_HI_KEY: The upper bound value of a histogram step
- RANGE_ROWS: The number of rows from the sample that fall within a histogram step, not counting the upper bound

QUEST SOFTWARE

DBCC SHOW_STATISTICS

- EQ_ROWS: The number of rows from the sample that are equal in value to the upper bound of the histogram step
- DISTINCT_RANGE_ROWS: The number of distinct values within a histogram step, not counting the upper bound
- AVG_RANGE_ROWS: The average number of duplicate values within a histogram step, not counting the upper bound (where RANGE_ROWS / DISTINCT_RANGE_ROWS for DISTINCT_RANGE_ROWS > 0)

QUEST SOFTWARE

TIP!

- Don’t forget to run UPDATE STATISTICS on a regular basis!
- The system stored procedure, sp_autostats, can be used to enable automatic statistic collection on an individual index, table, or all the tables in a table.

QUEST SOFTWARE

INDEX and TABLE FRAGMENTATION

- Table fragmentation is similar to hard disk fragmentation caused by frequent file creation, deletion and modification. Database tables and indexes need occasional defragmentation to stay efficient.
- The most efficient allocation for read-heavy tables is when all pages occupy a contiguous area in the database, but after weeks of use, a table may become scattered across the disk drive. The more pieces it is broken into – the less efficient the table becomes.

QUEST SOFTWARE

INDEX and TABLE FRAGMENTATION

- The most efficient allocation for write-heavy tables is when all pages occupy a contiguous area in the database, but have some unused space on each page (fill factor).
- The following DBCC commands help you maintain optimum table and index performance.

QUEST SOFTWARE

TIP!

- Dropping, recreating or reordering a clustered index recreates all other indexes on a table
- There are new and more efficient ways to recreate an index in one step rather than issuing separate DROP INDEX and CREATE INDEX statements (shown later)

QUEST SOFTWARE

DBCC SHOWCONTIG

- This command shows the degree of contiguous values in a clustered index. If the metrics are poor, then you can drop and recreate the clustered index. The syntax of this DBCC command is:

```
DBCC SHOWCONTIG [ ( object identifier [,
   index_name |
    index_id ] ) ]
   [WITH { ALL_INDEXES
          | FAST [, ALL_INDEXES ]
            | TABLERESULTS [, ALL_INDEXES]
                           [, {FAST |
   ALL_LEVELS} ] } ]
```

QUEST SOFTWARE

DBCC SHOWCONTIG

- You may use either table name and index name, or table ID and index ID numbers. For example:

```
USE northwind
GO
DBCC SHOWCONTIG ( [Order Details],
  OrderID )
GO
```

QUEST SOFTWARE

DBCC SHOWCONTIG

Results:

```
DBCC SHOWCONTIG scanning 'Order Details' table...
Table: 'Order Details' (325576198); index ID: 2, database ID:
   6
LEAF level scan performed.
- Pages Scanned.....................................: 5
- Extents Scanned...................................: 2
- Extent Switches...................................: 1
- Avg. Pages per Extent.............................: 2.5
- Scan Density [Best Count:Actual Count].....: 50.00% [1:2]
- Logical Scan Fragmentation ...................: 0.00%
- Extent Scan Fragmentation ....................: 50.00%
- Avg. Bytes Free per Page.....................: 2062.0
- Avg. Page Density (full).....................: 74.52%
DBCC execution completed. If DBCC printed error messages,
   contact your system administrator.
```

QUEST SOFTWARE

DBCC INDEXDEFRAG

- DBCC INDEXDEFRAG is a great way to rebuild the leaf level of index in one step
  - Performs on-line index reconstruction
  - Can be interrupted without losing the work already completed
  - Fully logged
  - Can take longer than rebuilding the index and is not quite as effective

Syntax:

```
DBCC INDEXDEFRAG ( { database | 0 } ,{ table | 'view' }
   ,{ index }    )
     [ WITH NO_INFOMSGS ]
```

QUEST SOFTWARE

DBCC DBREINDEX

- DBCC DBREINDEX was introduced in version 7.0 to enable DBAs to rebuild indexes without having to drop and recreate PRIMARY KEY and UNIQUE constraints
  - Locks the table for the duration of the operation
  - Can offer additional optimizations than a series of individual DROP INDEX and CREATE INDEX statements on a single table

Syntax:

```
DBCC DBREINDEX
      ( ['database.owner.table_name' [,index_name
  [,fillfactor] ] ]
    )     [ WITH NO_INFOMSGS ]
```

QUEST SOFTWARE

TIP!

- SQL Server 2000 now includes a new option on the CREATE INDEX statement, WITH DROP_EXISTING. This command offers a lot of benefits over the older technique of issue DROP INDEX statements followed by CREATE INDEX statements.
  - If executed on the clustered key, non-clustered keys are rebuilt
  - If the original columns and index names are used, the operation is sped up by not sorting the data again
  - Can be used to change the key(s) of an index

QUEST SOFTWARE

DBCC SQLPERF

- IOSTATS: Reports I/O usage since the server was started or since these statistics were cleared. The closer these values are to zero, the better.
- LRUSTATS: Reports cache usage since the server was started or since these statistics were cleared. LRU is Least Recently Used. Cache Hit Ratio is the single most important performance value in this group and indicates better results the closer it is to 100. (Similar to DBCC PROCCACHE)
- NETSTATS: Reports network usage.
- RASTATS: Reports Read Ahead usage.

QUEST SOFTWARE

DBCC SQLPERF

- CLEAR: This option is used in conjunction with one of the four discussed above. Clears the specified statistics and restarts generation of statistics. This option generates no output.
- THREADS: Maps the Windows NT system thread ID to a SQL Server spid. (Similar to sp_who).
- LOGSPACE: Reports the percentage of transaction log space used. This option can only be used if transaction log is located on its own database segment.

QUEST SOFTWARE

DBCC SQLPERF

Syntax:

```
DBCC SQLPERF (
   {IOSTATS   [, CLEAR]
   | LRUSTATS   [, CLEAR]
   | NETSTATS   [, CLEAR]
   | RASTATS [, CLEAR]
   | THREADS
   | LOGSPACE )
```

QUEST SOFTWARE

DBCC PROCCACHE

- DBCC, using the PROCCACHE option, can also be used to examine the procedure cache, that is, the space in memory reserved for caching the execution plans of stored procedures, triggers, functions, oft-called queries and so forth. The syntax is:

```
DBCC PROCCACHE
```

- You cannot directly tune the size of the procedure cache as you could in earlier versions of SQL Server. However, you can impact the size by controlling how much total memory SQL Server gets, and whether SQL Server must compete for memory with other Windows Services.

QUEST SOFTWARE

DBCC PROCCACHE

DBCC PROCCACHE returns the following information:

- num proc buffs: The number of stored procedures that could possibly be in the procedure cache
- num proc buffs used: The number of slots in the cache holding stored procedures. In this case, a slot is simply a position in the cache
- num proc buffs active: The number of slots in the cache holding stored procedures that are executing

QUEST SOFTWARE

DBCC PROCCACHE

- proc cache size: The total size of the procedure cache, in 8k pages
- proc cache used: The amount of the procedure cache holding stored procedures , in 8k pages
- proc cache active: The amount of the procedure cache holding stored procedures that are executing , in 8k pages

QUEST SOFTWARE

TIP!

- You can flush the procedure cache without rebooting the server by using the command DBCC FREEPROCCACHE. This command wipes clean all elements from the procedure cache.

Syntax:

```
DBCC FREEPROCCACHE
```

- Executing this command causes all compile plans in the procedure cache to be dropped.

QUEST SOFTWARE

DBCC PINTABLE

- Just as you can force clean the procedure cache, you can also force SQL Server to place data into the data cache. This can be dangerous since the data is never unpinned from memory. However, for the well-considered operation, DBCC PINTABLE can improve performance.

Syntax:

```
DBCC PINTABLE(database_id, table_id)
```

- The command does not actually read a table into the memory cache. Instead, it ensures that pages from the table are retained in cache once read. (You could couple this command immediately with a SELECT statement to read the table directly into memory.)

QUEST SOFTWARE

QUEST SOLUTIONS

- Spotlight® - Real-time diagnostic and resolution
- Foglight® - 24x7 unattended monitoring
- Knowledge Xpert™ - Online technical reference for SQL Server 2000
- QDesigner™ - Database design and application tool
- Benchmark Factory® - Load testing solution that scales throughput to virtually unlimited users

QUEST SOFTWARE

CONCLUSION

- Raffle giveaway
  - SQL in a Nutshell by Kevin Kline with Daniel Kline
- Survey/Questionnaire
- Upcoming e-Seminar
  - Analyzing and Optimizing T-SQL Query Performance on Microsoft SQL Server Part 2: Indexing Strategies
    - September 10, 2002
    - 11:00 a.m. - 12:00 p.m. PST
- Info@quest.com

QUEST SOFTWARE

QUESTION AND ANSWERS

QUEST SOFTWARE