SQLSERVER: Deframmentazione del database

Indice dei Contenuti [Nascondi/Mostra]

Articolo riportato da http://www.sql-server-performance.com/dt_dbcc_showcontig.asp

DBCC SHOWCONTIG is a wonderful tool which helps you to understand quite a bit more about your system than is obvious at first glance. And, frankly, the documentation doesn't use terminology that makes it very obvious either. So, this article will focus on a few of the big ideas behind the tool using SQL Server, and how you can use it to better understand what is going on inside your SQL Server box.

Probably one of the most significant performance problems found in databases is centered around table data fragmentation. One situation that may be analogous to table fragmentation might be an index at the end of a large book. A single index entry in such a book might point to several pages scattered throughout the book. You must then scan each page for the specific information you require. This differs significantly from the index of the phone book which stores its data in sorted order. A typical query for the name "Jones" might span multiple consecutive pages, but are always held in a sorted order.

In the case of a database, we start out with the data looking more like a phone book, and end with the data looking more like a history book. Therefore, we need to occasionally resort the data in an effort to recreate the phone book order. Below, you will see a graphical presentation of how SQL Server lays out the data so that we can discuss the actual findings more clearly.

A Quick SQL Server Internals Discussion

We are most familiar with the data row. The row size is set only by the definition of the table that holds it (e.g. A table of addresses require more data per row then a table of class names). In SQL Server, a table may define a row as storing as little as 4 bytes to as much as 8060.This limit is set by the size of the data page, which stores up to 8,192 bytes (8 KB). The remaining 132 bytes are used by SQL Server to track other information under the covers. Although SQL Server is designed around 8 KB pages, the smallest unit of data that SQL Server can allocate is 64 KB. This is called an extent.

To store the data in a sorted order, as in a phone book, SQL Server uses something called a clustered index. When a typical database is created, clustered indexes exist on nearly all tables. However, just because the data exists in sorted order within the page does not mean that it exists as such within an extent. The reason for this derives from situations in which there is no more room on a given page in which it can insert a row. SQL Server then removes approximately half the page and moves it to another page, which is called a Page Split (Page Splits will not occur with clustered indexes on IDENTITY based columns, but hotspotting may). In some cases, it may move that data to another extent altogether, possibly even allocating a new extent to do so. So, while we start off with names beginning with A and ending with H on one page, and names beginning with I and ending with Z on the next page, through usage, we may see that names A through C are now located on one page in one extent, D through E on another extent and S through Z back on the fifth page of the first extent, etc. It is because of the page split that there are times in which we may prefer to use tables with no clustered indexes at all. However, these tables are usually scratch tables which are highly volatile. In those situations, we desire the quicker write times at the cost of slower reads.

Calling DBCC SHOWCONTIG

Using Query Analyzer, connect to the database you wish to view. Next, you will need to get the object id of the table(s) you wish to examine. I have simplified this task to retrieve the top 10 tables by size using the following script.

```
SELECT TOP 10
'DBCC SHOWCONTIG(' + CAST(id AS NVARCHAR(20)) + ')'
+ CHAR(10) +
'PRINT '' ''' + CHAR(10)
FROM
sysindexes
WHERE
indid = 1 or
indid = 0
ORDER BY rows DESC
```

Execute this script in the database that you wish to check, and you will get an output resembling (repeated 10 times, once for each of the 10 largest tables):

```
DBCC SHOWCONTIG(123456789)
PRINT ''
```

Copy and paste the complete resultset into your query window and execute it.

The Results Explained

The results from the previous command will look something like the following:

```
DBCC SHOWCONTIG scanning 'MyTable1' table...
Table: 'MyTable1' (1556968673); index ID: 1, database ID: 16
TABLE level scan performed.
- Pages Scanned................................: 18986
- Extents Scanned..............................: 2443
- Extent Switches..............................: 9238
- Avg. Pages per Extent........................: 7.8
- Scan Density [Best Count:Actual Count].......: 25.70% [2374:9239]
- Logical Scan Fragmentation ..................: 44.58%
- Extent Scan Fragmentation ...................: 87.07%
- Avg. Bytes Free per Page.....................: 1658.7
- Avg. Page Density (full).....................: 79.51%
DBCC execution completed. If DBCC printed error messages,
contact your system administrator.


DBCC SHOWCONTIG scanning 'MyTable2' table...
Table: 'MyTable2' (183984032); index ID: 1, database ID: 16
TABLE level scan performed.
- Pages Scanned................................: 28980
- Extents Scanned..............................: 3687
- Extent Switches..............................: 22565
- Avg. Pages per Extent........................: 7.9
- Scan Density [Best Count:Actual Count].......: 16.06% [3623:22566]
- Logical Scan Fragmentation ..................: 83.05%
- Extent Scan Fragmentation ...................: 87.44%
- Avg. Bytes Free per Page.....................: 3151.1
- Avg. Page Density (full).....................: 61.07%
DBCC execution completed. If DBCC printed error messages,
contact your system administrator.
```

In the first table, MyTable1, we see that there were 18,986 pages examined to create the report. Those pages existed within 2,443 extents, indicating that the table consumed approximately 97% (7.8 pages per extent on average) of the extents allocated for it. We then see that while examining the pages for fragmentation, the server had to switch extent locations 9, 238 times. The Scan Density restates this by indicating the percentage of all pages within all extents were contiguous. In an ideal environment, the density displayed would be close to

100. The Logical Scan Fragmentation and Extent Scan Fragmentation are indications of how well the indexes are stored within the system when a clustered index is present (and should be ignored for tables that do not have a clustered index). In both cases, a number close to 0 is preferable. There is another anomaly being displayed here that is a little difficult to explain, but it is that SQL Server allows multiple tables to exist within a single extent, which further explains the 7.8 pages per extent (multiple tables may not however exist within a page).

The next items discuss a somewhat more mundane but important issue of page utilization. Again using the first table as the example, there are an average of 1659 bytes free per page, or that each page is 79.51% utilized. The closer that number gets to 100, the faster the database is able to read in records, since more records exist on a single page. However, this must be balanced with the cost of writing to the table. Since a page split will occur if a write is required on a page that is full, the overhead can be tremendous. This is exaggerated when using RAID 5 disk subsystems, since RAID 5 has a considerably slower write time compared to its read time. To account for this, we have the ability of telling SQL Server to leave each page a certain percentage full.

DBCC REINDEX is a related tool that will reorganize your database information in much the same way Norton Defrag will work on your hard drive (see Books Online for information on how to use DBCC REINDEX). The following report displays the differences in the data after we defragmented the data using DBCC DBREINDEX.

```
DBCC SHOWCONTIG scanning 'MyTable1' table...
Table: 'MyTable1' (1556968673); index ID: 1, database ID: 16
TABLE level scan performed.
- Pages Scanned................................: 15492
- Extents Scanned..............................: 1945
- Extent Switches..............................: 2363
- Avg. Pages per Extent........................: 8.0
- Scan Density [Best Count:Actual Count].......: 81.94% [1937:2364]
- Logical Scan Fragmentation ..................: 15.43%
- Extent Scan Fragmentation ...................: 20.15%
- Avg. Bytes Free per Page.....................: 159.8
- Avg. Page Density (full).....................: 98.03%
DBCC execution completed. If DBCC printed error messages,
contact your system administrator.

DBCC SHOWCONTIG scanning 'MyTable2' table...
Table: 'MyTable2' (183984032); index ID: 1, database ID: 16
TABLE level scan performed.
- Pages Scanned................................: 35270
- Extents Scanned..............................: 4415
- Extent Switches..............................: 4437
- Avg. Pages per Extent........................: 8.0
- Scan Density [Best Count:Actual Count].......: 99.35% [4409:4438]
- Logical Scan Fragmentation ..................: 0.11%
- Extent Scan Fragmentation ...................: 0.66%
- Avg. Bytes Free per Page.....................: 3940.1
- Avg. Page Density (full).....................: 51.32%
DBCC execution completed. If DBCC printed error messages,
contact your system administrator.
```

Here, we can see several key improvements and some examples of how proper indexing can be very important. The most glaring items for us are how well we were able to increase the scan density. Again, using the MyTable1 table as a reference, we can see that out of 1,945 extents, there were only 2363 extent switches. Notice that the number of extent switches is now a lower number than the original number of extents. This is due to the more efficient allocation of the data. And, since there is a significant reduction of the number of extent switches, searches for large quantities of contiguous data will be fulfilled much more quickly.

These reports were taken after only a small amount of processing had occurred on this system, yet already we can see that there has been a fair amount of fragmentation of the data. The table MyTable1 has already begun to show signs of performance degradation. When there is an unusually large amount of new data being inserted into the tables, these numbers will quickly begin to resemble the those that we see in the previous report.

In the table MyTable2, we see a stark difference from MyTable1.This is because of some index tuning that I had done on that table. As I said earlier, SQL Server uses the clustered indexes in order to understand how data should be ordered. To prevent page splits, I had SQL Server leave each page only 50% full. This allows for multiple inserts to occur without generating page splits, allowing our scan density to remain high for a longer period of time. But this also comes at the cost of reducing the quantity of contiguous records on each page and doubles the amount of space consumed by the table, hence the now much larger number of pages and extents scanned.

Conclusion

From examining the output of DBCC SHOWCONTIG, we were able to locate several key issues. First, we saw that our database was heavily fragmented, and required defragmentation using DBCC DBREINDEX. Next, we were able to tell what percentage of the allocated pages were actually being used by SQL Server. Finally, we saw that by modifying the fillfactor on an index, we had a tremendous affect on page splitting at the cost of more page I/O for each read.

Aggiornamento

L'utilizzo della funzione DBCC in sql server 2008 non è piu' supportato. Per fare la ebuild degli indici usare questa query che estrae e dati in ordine dal piu' frammentato.

```
SELECT 'alter index ' + i.name  + ' on ' +  OBJECT_NAME(i.OBJECT_ID)  + ' rebuild WITH (FILLFAC(?(
OBJECT_NAME(i.OBJECT_ID) AS TableName,
i.name AS IndexName,
indexstats.avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, NULL) indexstats
INNER JOIN sys.indexes i ON i.OBJECT_ID = indexstats.OBJECT_ID
AND i.index_id = indexstats.index_id
WHERE indexstats.avg_fragmentation_in_percent > 20
and i.name is not null
order by indexstats.avg_fragmentation_in_percent desc
```

Riferimenti

- http://blogs.technet.com/josebda/archive/2009/03/20/sql-server-2008-fragmentation.aspx
- http://www.sql-server-performance.com/articles/per/Analyze_and_Fix_Index_Fragmentation_in_SQL_Server_2008_p1.aspx
- http://www.sqlhacks.com/index.php/Optimize/Defragment-Data
- http://msdn.microsoft.com/it-it/library/ms175008.aspx
- http://www.databasejournal.com/features/mssql/print.php/2238211
- http://msdn.microsoft.com/it-it/library/ms177495.aspx
- http://msdn.microsoft.com/it-it/library/ms189858.aspx
- http://msdn.microsoft.com/it-it/library/ms188917.aspx
- http://msdn.microsoft.com/en-us/library/aa258828(SQL.80).aspx
- http://sqlfool.com/2009/06/index-defrag-script-v30/
- http://www.chrisneel.it/articolo/sqlindexoptim