UNISYS
imagine it. done.

white paper

Microsoft SQL Server 2008 Data and Backup Compression

Jerrold Buggert
Rick Freeman
Elena Shen
Richard Saunders
Cecil Reames

August 19, 2008

solutions

Table of Contents

Data compression of selected tables and indexes of a database and backup compression of database backups are two new features in Microsoft® SQL Server® 2008 database software that can be used in combination to reduce database and backup storage costs. Tables and indexes to be compressed generally should be chosen so as to reduce overall I/O rates and thus have a minimal impact on performance, unless spare CPU cycles are available. Backup compression is effective in general, even with an already compressed database, and reduces both storage and elapsed times for backup and restore. The CPU cost for backup compression is significant and can impact concurrent operations unless some means is used to limit backup CPU usage. Both new compression features should be considered, given appropriate circumstances where compression could be helpful.

Introduction to Compression in Microsoft SQL Server 2008

Microsoft SQL Server 2008 introduces two new compression features (available in Enterprise and Developer editions) that are of interest to users with large databases. Data compression applies to individual tables and indexes within a database and comes in two options: ROW and PAGE. Backup compression is a new option when backing up a database to external storage. Both compression techniques can be used to reduce the amount of online storage required for databases and their backups.

Data Compression

Data compression can be configured for an entire heap (table with no clustered index), clustered index, non-clustered index, or indexed view. For partitioned tables and indexes, the compression option can be configured separately (and differently) for each partition.[1]

The data compression option can be specified during CREATE TABLE or CREATE INDEX by using the new table_option DATA_COMPRESSION = NONE or ROW or PAGE. The data compression option can also be changed later through ALTER TABLE or ALTER INDEX with a (new for TABLE) REBUILD clause.

Data compression affects the physical storage of columns within a row and rows within a page on disk and in memory. It does not change the logical attributes of the data or the way it is presented by the database, so there are no changes visible to the application.[2]

Data compression requires more processing for select, insert, and update than for uncompressed data. Furthermore, compression is generally more expensive than decompression. For these reasons, in most cases, Microsoft recommends that compression *not* be used on tables and indexes if the size of the object is much smaller than the overall database or if the table is heavily used for DML operations.[3] Note that there is no database-wide compression option.

Compressing large tables and indexes that generate significant I/O volume can frequently improve their memory caching and reduce the I/O volume enough to compensate for (much of) the compression/decompression overhead, thus reducing storage costs without undue change in performance. In certain I/O-bound (or memory constrained) situations, data compression can even improve overall performance.

To estimate how changing the compression state will affect the amount of storage required for a table or index, use the *sp_estimate_data_compression_savings*[4] stored procedure or the Data Compression Wizard.[5]

ROW Compression

The new storage format for ROW compression works on each column in a row separately and has the following main changes:[2]

- It reduces metadata overhead associated with the record format.
- It uses variable-length storage for numeric types (e.g., integer, decimal, and float) and types based on numeric (e.g., datetime and money).
- It stores fixed character strings by using variable-length storage (not storing trailing blanks).

For numeric and decimal types, the storage format is identical to the vardecimal format of SQL Server 2005. For integer types, most significant value bytes with zeros are not stored. For floating-point types, least significant value bytes with zeros are not stored.

PAGE Compression

PAGE compression is a further extension of ROW compression. PAGE compressing the leaf level of tables and indexes consists of three operations performed in the following order:[6]

- Row compression
- Prefix compression
- Dictionary compression

Prefix compression looks for common leading byte patterns (regardless of data type) in each column across all rows in that page. Two or more instances of that byte pattern are stored in the column's location within an Anchor Record row, just after the page header.[7]

Dictionary compression looks for common byte patterns across all columns of all rows in the page. Multiple instances are stored in a Dictionary just after the Anchor Record, with the collective structure called Compression Information (CI). The repeated byte patterns are stored just once in the CI, and the columns in the multiple rows on the page merely reference those values.[8]

As rows are being added to a page, only ROW compression is performed. When the page becomes filled, PAGE compression is attempted. If there is enough free space left in the page after creating the CI, the PAGE compression is retained. Otherwise, the page is left with just ROW compression.

Non-leaf levels of indexes are compressed using only ROW compression.

Backup Compression

Backup compression is off by default. To change the server-level default, use the *sp_configure* stored procedure to set the value of configuration option 'backup compression default' to 1 and then execute the *reconfigure* statement.[9] To use SQL Server Management Studio, click the 'Compress Backup' checkbox on the 'Database Settings' section of the 'Server Properties' window, and this feature will be enabled on the server level. To override the backup compression default, use either WITH NO_COMPRESSION or WITH COMPRESSION in a BACKUP statement.[10]

Note that backup compression significantly increases CPU usage, and the additional CPU consumed by the compression process might adversely impact concurrent operations.

On the plus side, backup sizes and backup/restore elapsed times can be greatly reduced (depending on the compressibility of the database). Databases that use ROW and/or PAGE compression for most of their data will obviously be far less compressible for backups, but additional backup compression can occur.

Database Testing Considerations

The application and database chosen for this testing is representative of a modern OLTP financial environment. Both read-only and read-write transactions were implemented, ranging from relatively simple to fairly complex. The database consisted of small, medium, and large tables with primary and secondary indexes, foreign keys, data checks, and referential integrity checks. The columns spanned a wide variety of data types. The rows were initially populated with realistic data values, so compression results should also be realistic.

Database Information

The database tables and indexes were broken into related categories and stored in separate filegroups. The number of files in each filegroup was the same as the total number of data volumes, with each data volume containing a separate volume partition for each filegroup. The total data size was about 2 TB.

Care was taken to pick appropriate FILLFACTOR values for certain tables and indexes in the database to reduce potential page splits and fragmentation during run time.

Selection of Tables/Indexes to Be Compressed

Following Microsoft recommendations, the following stored procedure was executed to determine the best compression opportunities: [11]

sp_estimate_data_compression_savings

In addition, data from executing the uncompressed database was gathered to give an idea of which tables and indexes caused the largest amount of physical reads to disk, used the largest number of buffers in the buffer pool, and had the smallest percentage of database data cached in memory. Consult SQL Server 2008 Books Online for more information on using objects like the following: [12]

::fn_virtualfilestats

sys.dm_os_buffer_descriptors

sys.master_files

sys.allocation_units

sys.dm_index_operational_stats

sys.dm_index_physical_stats

sys.dm_exec_cached_plans

sys.dm_exec_sql_text

Selection of ROW vs. PAGE Compression

Having selected the tables and indexes that might benefit the most from compression, the next task is to choose which type of compression to use: ROW or PAGE. Since PAGE compression starts with ROW compression on the individual rows, and then further compresses repeated data patterns across the entire page, it obviously requires more processing for both compression and decompression. Will the extra space savings justify the larger overhead? Will the reduction in I/O be enough to compensate? Or is the reduction in storage most important? [13]

Disk Storage & I/O Reduction vs. Performance

When either type of compression is used, there is a multi-way trade-off involved between storage space (disk and buffer pool), I/O reduction (due to better memory caching), and performance (loss due to compression/decompression overhead, but gain due to lower I/O rate). The results below show that it is possible in some cases to achieve an overall win in all areas, but doing so isn't easy and certainly isn't possible in all cases.

Test Configuration

The system used for compression testing was the Unisys® ES7000™/one Enterprise Server. Configuration details are explained in the following sections.

Database Server

The ES7000/one Enterprise Server was configured with one partition containing 16 Intel® Dual-Core Xeon™ 7140M processors (32 processing cores). Hyperthreading was disabled. The following table shows a summary of the database server components.

Table 1: Database server components

System Component	Description
Processors	Intel® Dual-Core Xeon™ 7140M CPU 3.40GHz, 2 Core(s) (2 Logical Processor) w/16-MB L3 cache
Memory	128 GB RAM
Network interface card	8 integrated Ethernet ports (only 4 ports were used)
Network protocol	TCP/IP
Disk HBA	7 Emulex LightPulse LP11000 dual port
Disk subsystem for the database	13 RTS 4200 RAID Controllers 26 LUNs RAID 10, 22 disks per LUN, 73 GB per disk, 15k RPM
Disk subsystem for the database log	1 RTS 4200 RAID Controller 1 LUN RAID 10, 6 disks per LUN, 146 GB per disk, 15k RPM
Operating system	Microsoft Windows Server ® 2008 Datacenter Edition, Version 6.0.6001 Service Pack 1 Build 6001
Database manager	Microsoft SQL Server 2008 Enterprise Edition (64 bit), RC0 build

I/O Subsystem

The storage subsystem consisted of two types of disk array systems. Database data consisted of 572x73-GB disks configured as 26x22-disk RAID10 data arrays. Six 146-GB disks were configured as one RAID10 Database log array.

Fourteen RTS 4200 RAID Controllers were connected to the seven dual-port FC disk adapters installed in the database server. Each of thirteen RTS 4200 RAID controllers managed two 22-disk RAID-10 data arrays. One RTS 4200 RAID controller was used to manage the log array.

Microsoft SQL Server 2008

All SQL Server configuration parameters were unchanged from their installation default values for these tests. No trace flags were used.

Compression Results

The results for data compression are given first, followed by those for backup compression.

Data Compression

The following data compression tests were performed:

Test	ROW Compression	PAGE Compression
1	none	none
2	Large tables and indexes	
3	NC indexes of large tables	Large tables
4		Large tables and indexes
5		All tables and indexes

For tests 2-4, only the large tables and indexes, which made up the overwhelming majority of the database size, were selected for data compression, while the small and medium tables were not compressed. For test 5, PAGE compression was applied to all the tables and indexes in the database (including small and medium tables that are completely cached in the buffer pool even when not compressed).

After building the uncompressed database, *sp_spaceused* was executed to determine the initial uncompressed database size. For each compression test, the following additional steps were performed:

1. Restore the uncompressed database.
2. Rebuild the tables and indexes with PAGE or ROW compression by executing the appropriate ALTER statement for each table or index:

   ALTER TABLE <table_name> REBUILD WITH (DATA_COMPRESSION=PAGE);

   ALTER TABLE <table_name> REBUILD WITH (DATA_COMPRESSION=ROW);

   ALTER INDEX <index_name> REBUILD WITH (DATA_COMPRESSION=PAGE);

   ALTER INDEX <index_name> REBUILD WITH (DATA_COMPRESSION=ROW);

   Then execute *sp_spaceused* to calculate the database size for the initial compressed database.
3. Make a performance run; then execute *sp_spaceused* again to calculate the run-time database size.

Space Reduction

The compression ratio is defined as follows:

Compression Ratio = Uncompressed Size / Compressed Size

Not surprisingly, the compression tests show PAGE compression on the large tables yields the best compression ratio and greatest space reduction for the initial database.

Table 2: Comparing initial data compression ratios and space savings

Test	Data Compression	Initial Data Compression Ratio	Initial Database Space Savings
1	Uncompressed	1.00	0%
2	ROW	1.21	18%
3	PAGE&ROW	1.42	30%
4	PAGE	1.85	46%
5	PAGE-All-Tables	1.86	46%

The following table shows the compression ratios and storage space savings differentials after running approximately the same total number of transactions on the initial database for each test.

Table 3: Comparing run-time data compression ratios and space savings

Test	Data Compression	Run Time Compression Ratio	Run Time Space Savings
1	Uncompressed	1.00	0%
2	ROW	1.06	6%
3	PAGE&ROW	1.15	13%
4	PAGE	1.42	30%
5	PAGE-All-Tables	1.41	29%

Note that run-time space is determined by first subtracting total database space after a run from initial database space before the run, next dividing by the total number of new transactions executed, and then computing the normalized ratios by dividing the test 1 result by each of the test results. Run-time results are lower than initial due to some page fragmentation, with the main factors being newly allocated pages are only partially filled and some filled pages later get split due to out-of-key-order insertion or update of a variable-length column with a larger data value.

I/O Reduction

As expected, the compression tests also show PAGE compression achieves highest I/O reduction.

Table 4: Comparing disk I/O ratios on full, half, and quarter memory configurations

Test	Compression	Full Memory	Half Memory	Quarter Memory
1	Uncompressed	1.00	1.43	2.16
2	ROW	0.92	1.15	2.13
3	PAGE&ROW	0.87	1.12	2.14
4	PAGE	0.78	0.98	1.34
5	PAGE-All-Tables	0.72	0.91	1.22

Note that the smaller memory configurations show a much bigger I/O reduction than does the full memory configuration. In other words, compression improves caching in a smaller buffer pool more than in a larger buffer pool. Note that the first three tests for quarter memory were I/O bound (see Table 6 below), which is why they show such large relative I/O rates and such large improvements with PAGE compression.

Performance Impact

Except for test 5, performance improved due to lower I/O rates and improvement in memory utilization. Note that the effect on performance was much more dramatic with less memory.

Table 5: Comparing performance ratios on full, half, and quarter memory configurations

Test	Compression	Full Memory	Half Memory	Quarter Memory
1	Uncompressed	1.00	0.89	0.34
2	ROW	1.01	0.96	0.44
3	PAGE&ROW	1.01	0.96	0.45
4	PAGE	1.02	0.97	0.92
5	PAGE-All-Tables	0.93	0.90	0.86

Note that in test 5, compressing all the tables and indexes in the database had a negative impact on performance for all three memory configurations. Also note that with PAGE compression, the half memory performance got within 3% of the full memory performance of the uncompressed database, and the PAGE compressed quarter memory performance actually was 3% better than the half memory uncompressed performance. These results show that at least some I/O bound situations have the potential for dramatic gains through use of data compression.

The following table shows that CPU utilization was almost fully busy for all tests and memory configurations, except for the first three tests with only quarter memory.

Table 6: Comparing average CPU % on full, half, and quarter memory configurations

Test	Compression	Full Memory	Half Memory	Quarter Memory
1	Uncompressed	100%	100%	81%
2	ROW	100%	100%	91%
3	PAGE&ROW	100%	100%	93%
4	PAGE	100%	100%	100%
5	PAGE-All-Tables	100%	100%	100%

Note that the system was I/O bound during the first three tests of quarter memory (i.e., the CPUs were not 100% busy because they were often stalled waiting for I/Os to complete). PAGE compressing the database alleviated that situation by reducing the I/O volume and thus greatly improving the performance.

Index Rebuild Time

The time required to rebuild the tables/indexes is significantly higher with PAGE compression than it is for ROW compression, which is indicative of the extra compression effort expended for PAGE. The ratio shown below is the elapsed time for rebuilding each test divided by the elapsed time for test 1.

Table 7: Comparing tables/indexes rebuild time

Test	Data Compression	Ratio
1	Uncompressed	1.00
2	ROW compressed	1.41
4	PAGE compressed	2.40

Note that compression can also be applied at CREATE TABLE or CREATE INDEX time, but the compression time is still quite significant.

Backup Compression

The following test results were performed on the initial database with all three levels of database data compression (NONE, ROW, and PAGE) and with both combinations of backup compression (NO_COMPRESSION and COMPRESSION).

After a compressed backup, execute the following statement to get the backup compression ratio:

```
SELECT backup_size/compressed_backup_size
FROM msdb.dbo.backupset
```

The backup compression results are broken into several areas: storage space reduction, and elapsed time reduction versus CPU increase.

Storage Space Reduction

The backup compression storage space savings for the uncompressed initial database is almost twice as much as the backup compression savings for the PAGE compressed initial database, which is to be expected, given that the latter database is already compressed. The good news is that backup compression can further reduce the size of an already compressed database. That isn't too surprising, however, since the goal of data compression in the database isn't maximum compression, but a balance of compression with minimum overhead for compressing and decompressing.

Table 8: Space reduction for compressed versus uncompressed backups

Data Compression	Backup Compression Ratio	Space Reduction
Uncompressed	2.95	66%
ROW compressed	2.42	59%
PAGE compressed	1.57	36%

Note that the compressed backup size is about the same in all three cases (but slightly larger the more highly compressed the database is).

Elapsed Time Reduction vs. CPU Increase

The elapsed time required to make compressed backups can be significantly less because the size of compressed backups are smaller and there are fewer writes to the backup media. However, overall CPU is significantly higher because of the compression overhead.[14]

CPU-Seconds is defined as following:

CPU-Seconds = 'backup elapsed time' x 'avg CPU %' x 'number of CPUs'

The reason CPU-Seconds is defined this way is an attempt to provide a metric that will not depend as much on the particular database, backup file, disk configuration, and number of processors.

Table 9: Uncompressed backup: elapsed time, CPU %, CPU-Seconds, and CPU-Seconds per GB

Data Compression	DB Size in GB	Elapsed Time (sec)	Avg. CPU %	CPU-Seconds	CPU-Seconds per GB
Uncompressed	2,080	6,937	1.42	3,155	1.5
ROW compressed	1,716	4,125	2.44	3,225	1.9
PAGE compressed	1,123	2,581	1.97	1,630	1.5

Table 10: Compressed backup: elapsed time, CPU %, CPU-Seconds, and CPU-Seconds per GB

Data Compression	DB Size in GB	Elapsed Time (sec)	Avg. CPU %	CPU-Seconds	CPU-Seconds per GB
Uncompressed	2,080	2,358	77.26	58,285	28.0
ROW compressed	1,716	2,188	76.75	53,745	31.3
PAGE compressed	1,123	1,512	86.55	41,886	37.3

Microsoft warns that backup compression can significantly increase CPU utilization and that it may have an adverse impact on concurrent operations. They suggest that Resource Governor[15] can be used to limit the CPU usage of a compressed backup session when CPU contention occurs.[9] However, because backup compression is highly parallelized, the more CPU resources that are available, the more the backup elapsed time can be reduced.

Summary and Recommendations

Both data compression and backup compression are highly useful new features added to SQL Server 2008.

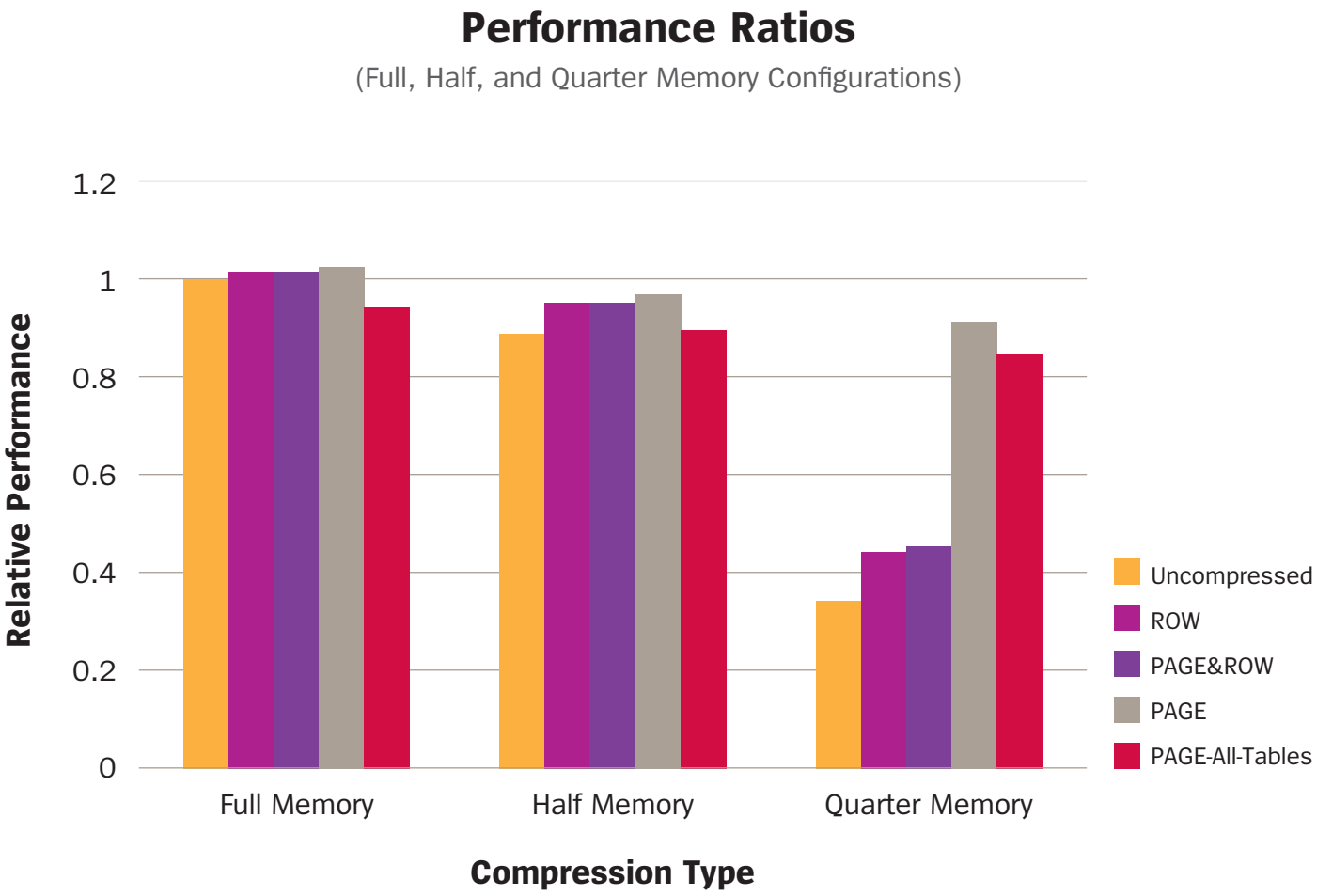
Backup Compression

Of the two, backup compression takes far less effort to start using: just add WITH COMPRESSION to the BACKUP DATABASE statement (or change the server configuration default). If backups are performed concurrently with other database operations, the system may be impacted if something isn't done first to limit the CPU used by the backup. The backup CPU usage does increase very significantly due to the cost of compression. Aside from that consideration, however, using backup compression is recommended to reduce both backup storage and elapsed time.

Database Compression

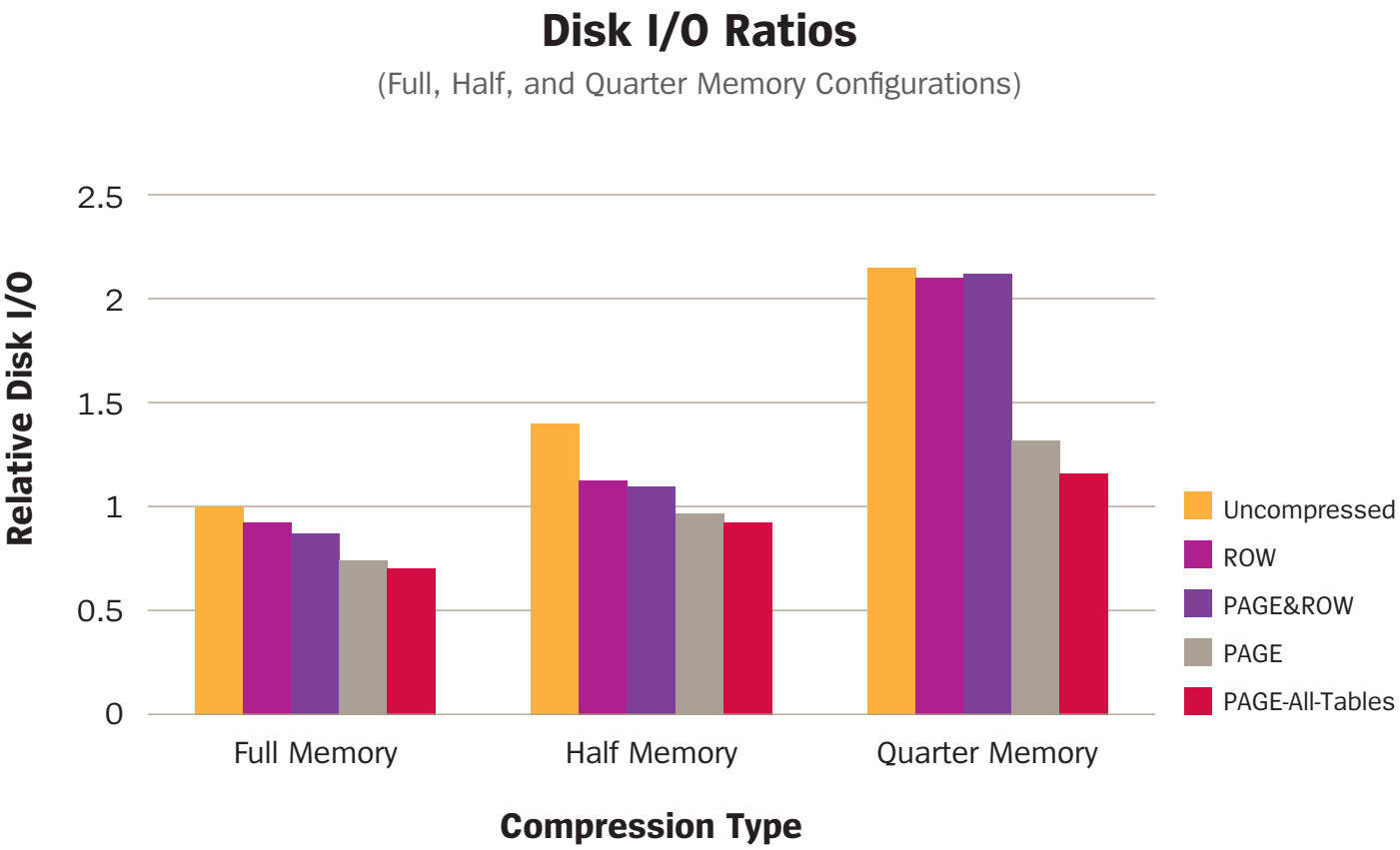
Recommending how to use data compression properly is a lot more complicated. There is no database-wide data compression option, and rightly so. Compression must be specified for individual tables and indexes. Picking which tables and indexes to compress (and which of two different compression options to use) is a difficult decision. Most importantly, there are tradeoffs that must be made depending on the major objective.

The graph below shows that in situations where the system memory is insufficient for a large enough buffer pool, resulting in an I/O bound configuration that cannot keep the processors completely busy, being able to PAGE compress the database can improve performance dramatically. Note, of course, that larger memory configurations see less of a gain, and that actual results depend on many other factors.



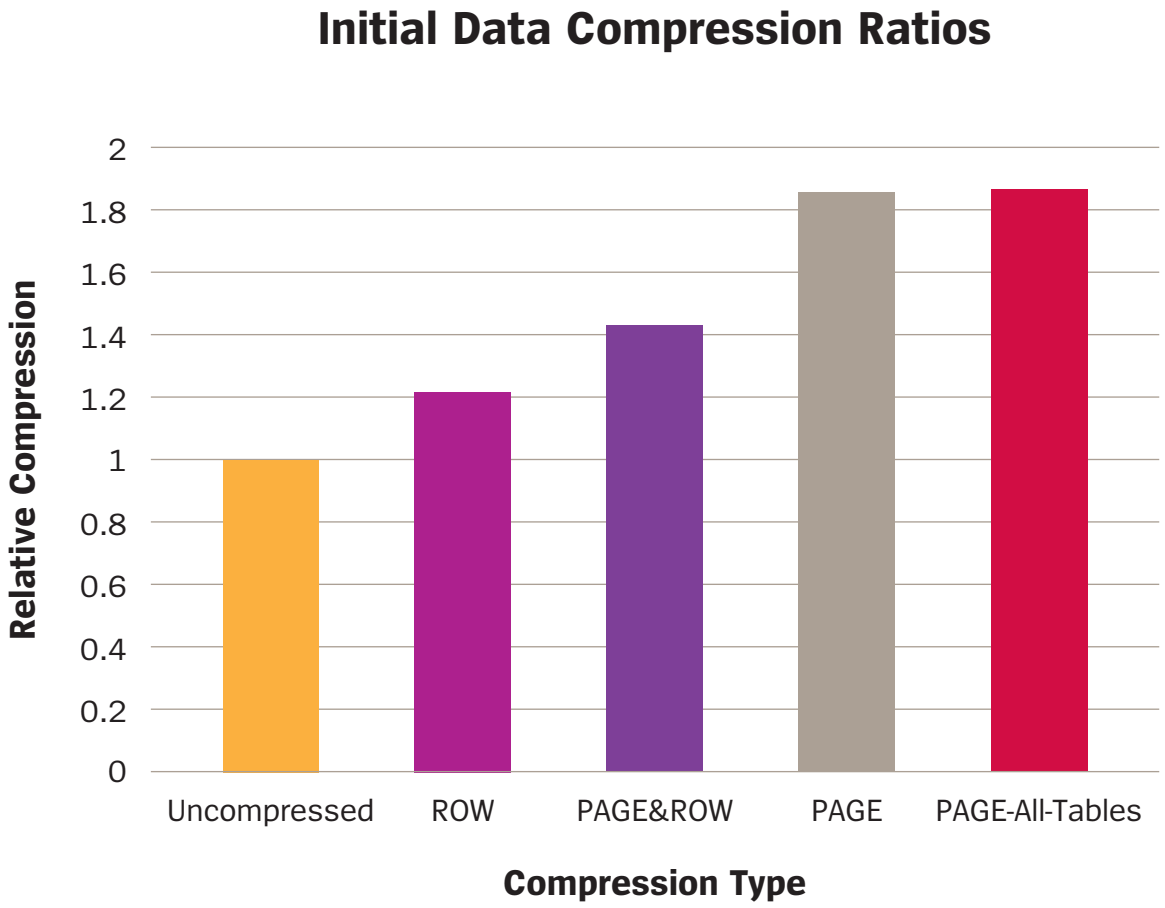
In general, compressing tables and indexes that fit comfortably within the buffer pool is likely to hurt performance and may do little to reduce storage needs, if most of the storage is consumed by large tables. Compressing just the largest tables (and their most used indexes) is likely to produce most of the storage reductions and may reduce the I/O rate enough to mitigate the cost of compressing new data and decompressing all data as it is accessed from the buffer pool.

If memory was insufficient initially, the I/O savings can even be large enough to improve performance, as shown in the following graph. Note that these trade-offs will likely be different for each system and database.



For systems with spare CPU cycles for performing compression, the objective may be just reducing the amount of online storage needed for holding the database. The following graph shows how much compression can reduce the amount of disk storage needed. (Note that disks needed to maintain the I/O rate is another consideration.)

If there aren't many spare CPU cycles, then picking which compression option to use is harder. ROW compression provides variable-length storage of values for fixed-length columns at relatively low compression cost, but it also yields less compression. PAGE compression starts out with ROW compression and then looks for repeated values across the rows of the entire page. The repeated values can then be stored just once in a page dictionary and referenced from multiple rows and columns within the page. The actual PAGE compression is delayed until the page fills completely, but it is still more expensive. PAGE decompression is also somewhat more expensive than for ROW. So again, there are CPU costs versus disk space versus I/O rate tradeoffs to be made.



Considering one final situation, if the database is partitioned, and data in older partitions is accessed much less frequently, then deciding to use ROW and then PAGE compression just on older partitions, to reduce online storage for older data, might be an easy decision to make.

In summary, consider using SQL Server 2008 data compression for appropriate situations, based on a thorough examination of all factors, the various tradeoffs, and the particular objective to be met. The storage reductions and/or performance improvements for I/O bound systems can be sizeable, so the potential payoff can justify the significant initial effort required. Recommending the use of backup compression is much easier. It requires little effort to get started, although limiting its significant CPU usage is advised if backup is performed along with concurrent database operations. Both compression features can be used together.

References

1. SQL Server 2008 Books Online. Topic: “Creating Compressed Tables and Indexes”
2. SQL Server 2008 Books Online. Topic: “Row Compression Implementation”
3. http://blogs.msdn.com/search/SearchResults.aspx?q=compression+AND+user%3aSunil&o=Relevance
   SQL Server Storage Engine blog, Sunil Agarwal, “Compression Strategies”
4. SQL Server 2008 Books Online. Topic: “sp_estimate_data_compression_savings (Transact-SQL)”
5. SQL Server 2008 Books Online. Topic: “Data Compression Wizard F1 Help”
6. SQL Server 2008 Books Online. Topic: “Page Compression Implementation”
7. http://blogs.msdn.com/search/SearchResults.aspx?q=compression+AND+user%3aSunil&o=Relevance
   SQL Server Storage Engine blog, Sunil Agarwal, “Details on PAGE compression (column-prefix)”
8. http://blogs.msdn.com/search/SearchResults.aspx?q=compression+AND+user%3aSunil&o=Relevance
   SQL Server Storage Engine blog, Sunil Agarwal, “Details on PAGE compression (page-dictionary)”
9. SQL Server 2008 Books Online. Topic: “Backup Compression (SQL Server)”
10. SQL Server 2008 Books Online. Topic: “BACKUP (Transact-SQL)”
11. http://blogs.msdn.com/search/SearchResults.aspx?q=compression+AND+user%3aSunil&o=Relevance
    SQL Server Storage Engine blog, Sunil Agarwal, “Estimating the space savings with data compression”
12. http://sqlcat.com/toolbox/archive/2008/02/21/scripts-and-tools-for-performance-tuning-and-troubleshooting-sql-server-2005.aspx
13. http://blogs.msdn.com/search/SearchResults.aspx?q=compression+AND+user%3aSunil&o=Relevance
    SQL Server Storage Engine blog, Sunil Agarwal, “Data compression techniques and trade offs”
14. http://blogs.msdn.com/sqlcat/archive/2008/03/02/backup-more-than-1gb-per-second-using-sql2008-backup-compression.aspx
    SQL Server Development Customer Advisory Team, Lindsey Allen & Thomas Grohser, “Backup More Than 1GB per Second Using SQL2008 Backup Compression”
15. SQL Server 2008 Books Online. Topic: “How to: Use Resource Governor to Limit CPU Usage by Backup Compression (Transact-SQL)”

August 2008

4137 1394-000