Assessing the Impact of Compression and Encryption on Microsoft® SQL Server® 2008 Databases

Database Solutions Engineering

By Kevin Guinn; with Bryant Vo and Leena Kushwaha

Dell Product Group
April 2009

Executive Summary

Microsoft SQL Server 2008 Enterprise Edition offers data compression and encryption features that save space and provide data security. Before deploying encryption or compression, however, you should understand how they will affect your database. There are likely to be important tradeoffs, and the impact of using each feature will vary based on the nature of your data and the types of workloads for which the database is designed. For example, both the compression and encryption features are implemented in software, so they increase processor utilization. Also, compression impacts the I/O characteristics of a database workload.

To provide perspective and to enable database administrators to make informed decisions, Dell ran several experiments that simulated common transaction processing and decision-support database workloads. We used the Dell DVD Store transactional database and a car sales data warehousing database to analyze the impacts of compression and encryption. This white paper provides the results of these laboratory tests, and offers insight into the trade offs and potential applicability of these features to various database workloads.

Table of Contents

Introduction

System utilization and security are important factors when deploying a data-centric business application. Microsoft SQL Server 2008 Enterprise Edition includes data compression and encryption features that can help address these challenges. The native data compression operations are applied to a table or index, and two different methods—row and page compression—are available to reduce the size of the data. Encryption operations are applied to a database to protect information stored in the data files.

The compression techniques used in SQL Server 2008 are software-based, and workloads that employ compressed tables are therefore expected to consume more CPU resources than if the tables were not compressed. Additionally, compression should alter not only the size of the data on disk, but also the nature of the I/O while the host processes a workload. Similarly, the native encryption operations are implemented in software, so the database server is expected to consume additional CPU resources when this feature is implemented. To discern the nature of these changes, Dell's engineers monitored SQL Server hosts as they ran different representative workloads.

Data Compression

The SQL Server 2008 data engine offers two different levels of data compression: row compression and page compression. Row compression saves space by storing all fixed-length data types as if they were variable-length. Fixed-length data types include those used for character, integer, floating-point, and other numeric-based data. SQL Server offers several different integer data types, as outlined in Table 1. If the range of possible values in a column is known when the database is designed, a data type that requires less storage space can be used. Unfortunately, it is not always possible to foresee exactly what data will eventually be stored, and the larger data types often offer greater flexibility to database and application designers and administrators. Row compression can preserve this flexibility while reducing the required storage space for a database.

Table 1: SQL Server Integer Data Types

Data Type	Allowed Values	Required Storage Space
bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8 Bytes
int	-2,147,483,648 to 2,147,483,647	4 Bytes
smallint	-32,768 to 32,767	2 Bytes
tinyint	0 to 255	1 Byte

Consider a table that contains an ***int*** column in which common values are smaller than 255. Without the benefits of compression, each entity in this column requires four bytes of storage. With row compression, those values can be stored in a variable-length format and will each require just one byte of storage instead of four. Similarly, if a ***char(50)*** data type is designated to store names, but the typical name stored is ten or fewer characters, there could be significant savings (80% in this example). For more detailed information, see Row Compression Implementation in SQL Server 2008 Books Online.

Page compression first applies Row compression, and then uses two additional techniques to identify commonalities among the information stored in a data page. These commonalities are used to further reduce the required storage space. The first additional technique is called prefix compression— it works by looking for common data within a column. The second additional technique is called dictionary compression—it optimizes commonalities regardless of where they occur within the data page. As a result, page compression should offer

greater space savings than row compression. For more detailed information, see Page Compression Implementation in SQL Server 2008 Books Online.

Both types of compression can be applied to an individual table or index, or – if applicable – to a partition of one of these entities. The *sp_estimate_data_compression_savings* stored procedure or the *Data Compression Wizard* in SQL Server Management Studio can be used to determine how much space will be saved by implementing data compression.

Transparent Data Encryption

Transparent Data Encryption (TDE) derives its name from the fact that the encryption occurs in a manner that does not require applications or queries to be aware of the fact that the data is encrypted. One reason this is possible is that the data is only encrypted when it is "at rest" and stored on disk. Data in memory and data in transit—for example, between the SQL Server host and an application server—are not encrypted using this feature. In a secure application infrastructure, TDE compliments traditional transport-level encryption techniques, such as Secure Sockets Layer (SSL) and Transport Layer Security (TLS).

TDE is applied to a database, and requires that a set of certificates and keys be created. A database encryption key is necessary to allow encryption and decryption of a user database. The key should be backed up and stored independently. The key is required not only to access stored data, but also to restore encrypted data that has been backed up. While only SQL Server Enterprise Edition can perform encryption, any edition has the capability to restore encrypted backups (provided that the key is available). For more detailed information, see Understanding Transparent Data Encryption on SQL Server 2008 Books Online.

Transaction Processing Results

The Dell DVD Store version 2 (DS2) database (http://linux.dell.com/dvdstore/) is designed to simulate an e-commerce operation, and was selected to represent a transaction processing workload. The DS2 build scripts were used to create a "large" dataset, which is nominally 100GB. The SQL Server driver application was used to simulate customers interacting with an e-commerce site. These interactions generate connections to the DS2 database, and cause transactions to be executed as customers search for and order movies from the online store.

A PowerEdge R710 with a PERC 6/E and a PowerVault MD1120 was used to house the DS2 database. The server was running 64-bit extended (x64) versions of Windows Server 2008 and SQL Server 2008, and was configured with a single quad-core processor and 4 GB of RAM. Following best practices, separate volumes on the MD1120 were allocated for data files, transaction logs, and TempDB. Each of these volumes used RAID 1/0; four physical disks were allocated for logs, four for TempDB, and twelve for data files. Four separate data files (one per physical processing core) were used to enable SQL Server I/O operations to be completed in parallel.

To determine the effects of encryption and compression on the SQL Server 2008 host, the workload described above was repeated as these features were applied to the DS2 database or to its tables and indexes. Several iterations were run in each of six configurations, which are outlined in Table 2. The T-SQL scripts that were used to implement compression and encryption are included in Appendix A and Appendix B, respectively.

Table 2: Test Configurations

Configuration	Description
Baseline	Neither compression nor encryption is applied to the database or its tables and indexes.
Row Compression	Row compression was applied to all of the DS2 tables (except for the REORDER table, which is empty at the beginning of the workload). Compressing the table automatically compresses the first clustered index associated with the table. Additionally, the IX_PROD_SPECIAL_CATEGORY_PRODID index of the PRODUCTS table was compressed.
Page Compression	Page compression was applied to the same tables and indexes as above.
Transparent Data Encryption (TDE)	The DS2 database was encrypted, but no compression options were used.
TDE and Row Compression	The DS2 database was encrypted, and the tables and indexes were compressed as described for Row compression above.
TDE and Page Compression	The DS2 database was encrypted, and the tables and indexes were compressed as described for Page compression above.

Disk Subsystem Throughput and Bandwidth

For transaction processing workloads, the throughput of the disk subsystem (as measured in Input/Output operations per second, or IOPS) is often a critical constraint. While it is not as likely to be a constraint, the bandwidth (as measured in megabytes per second) is also of interest. As the workload was applied to each configuration in Table 2, we recorded both of these criteria. The results of these tests are summarized in Table 3.

Table 3: Disk Throughput with the Dell DVD Store Database

Configuration	Mean Reads/s	Median Reads/s	Maximum Reads/s	Mean Writes/s	Median Writes/s	Maximum Writes/s
Base	168.90	178.95	218.76	131.22	25.93	487.84
Row Compression	146.63	139.40	336.05	96.92	20.20	393.64
Page Compression	109.01	109.33	142.99	86.99	17.00	324.38
TDE	151.50	151.28	173.00	121.78	26.74	456.13
TDE + Row	122.86	122.39	562.42	82.88	20.67	581.21
TDE + Page	92.77	92.34	140.80	76.14	18.33	340.07

The DVD Store workload used for these tests did not place the I/O subsystem under significant stress; the observed disk latencies were all under 10ms, which is well below the maximum threshold that is acceptable for a transactional database. Similarly, there was adequate overhead for the disk subsystem to process additional I/O based on the capabilities of the PERC 6/E and MD1120.

Under these conditions, the effects of compression followed a similar pattern regardless of whether encryption was enabled or not. As expected, less data was transferred to/from the database volumes, and fewer I/O operations were required to satisfy the needs of the workload. As the level of compression increased, this trend continued, indicating that the reduction in IOPS that are required to serve the same workload could reduce the

number of spindles that serve a given database. This also indicates that page compression could be implemented as part of a solution that consolidates several less-critical databases onto fewer SQL Server hosts.

The raw disk performance appears to have been reduced when the database was encrypted. With TDE enabled, each data page that is written to the physical disk is encrypted, and each page that is read is decrypted. These operations require a discrete amount of time to complete, and therefore may serve to slightly reduce the observed performance of the disk subsystem. Depending on whether compression was also enabled, the reduction ranged from 10% to 20% relative to the test configurations where TDE was not enabled.

Disk Space Used by the Database and Indexes

The primary reason to deploy data compression is to reduce the space required to store the database. Figure 1 shows the space that was used for the DS2 data and indexes for each tested configuration.

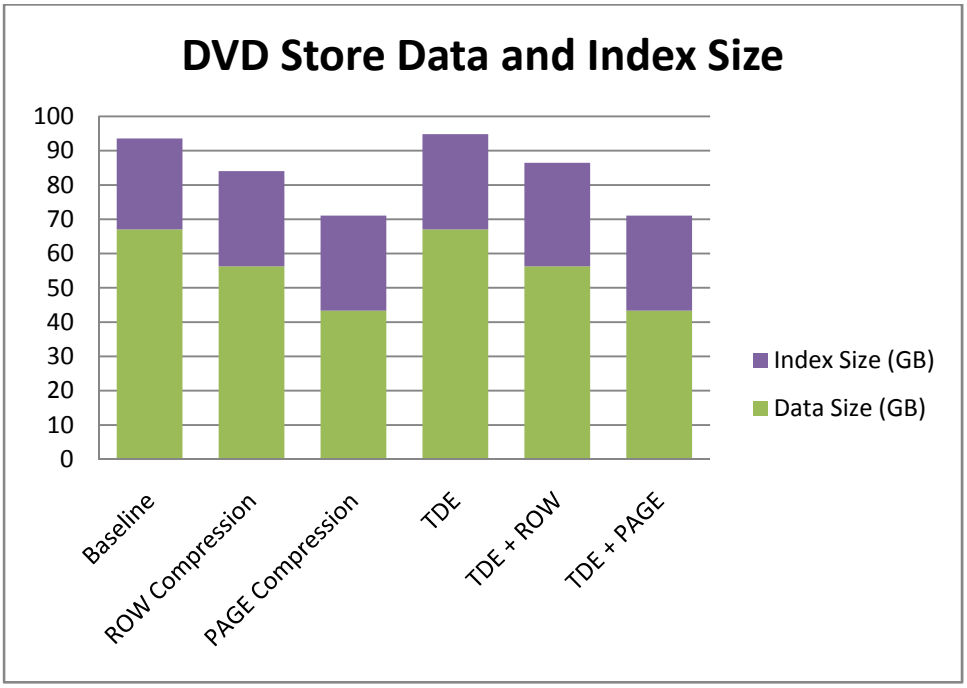

Figure 1: Disk Space Used by the Dell DVD Store Database

Most of the character data in the DS2 database is already stored using the ***varchar*** data type, and many of the integer columns employ ***smallint*** and ***tinyint***. However, some of the character data in the DS2 database is randomly generated, and therefore not very compressible. Even with this constraint, row compression offered a 16% reduction in the size of the data, and page compression offered a 35% reduction compared to the baseline. Additionally, the primary clustered indexes were not compressible, and therefore the space required to store the indexes grew slightly. Regardless of this phenomenon, the aggregate space savings are clearly visible in Figure 1, and these trends are the same irrespective of whether the database has been encrypted.

Because TDE encrypts each data page as it is written to disk, the storage space required is not expected to change when encryption is enabled. These results verify this hypothesis, with negligible increases (less than 2/100 of a percent) being observed between encrypted and unencrypted configurations with the same level of compression.

Processor Utilization

In order to compress and uncompress data, the processors in a SQL Server host are expected to perform additional work. This same expectation holds for encryption and decryption operations. Figure 2 shows the observed CPU utilization from the SQL Server host. For this set of test configurations, the R710 was configured with a single quad-core processor and had 4 GB of physical memory. In addition, Hyper-threading was enabled, so Windows Server 2008 reported eight logical processors.

The baseline DS2 test workload stabilized after the initial ramp-up period, and consumed approximately 43% of the host's processor time for the duration of the test. This indicates that the system was capable of handling the workload without experiencing contention for processor resources.

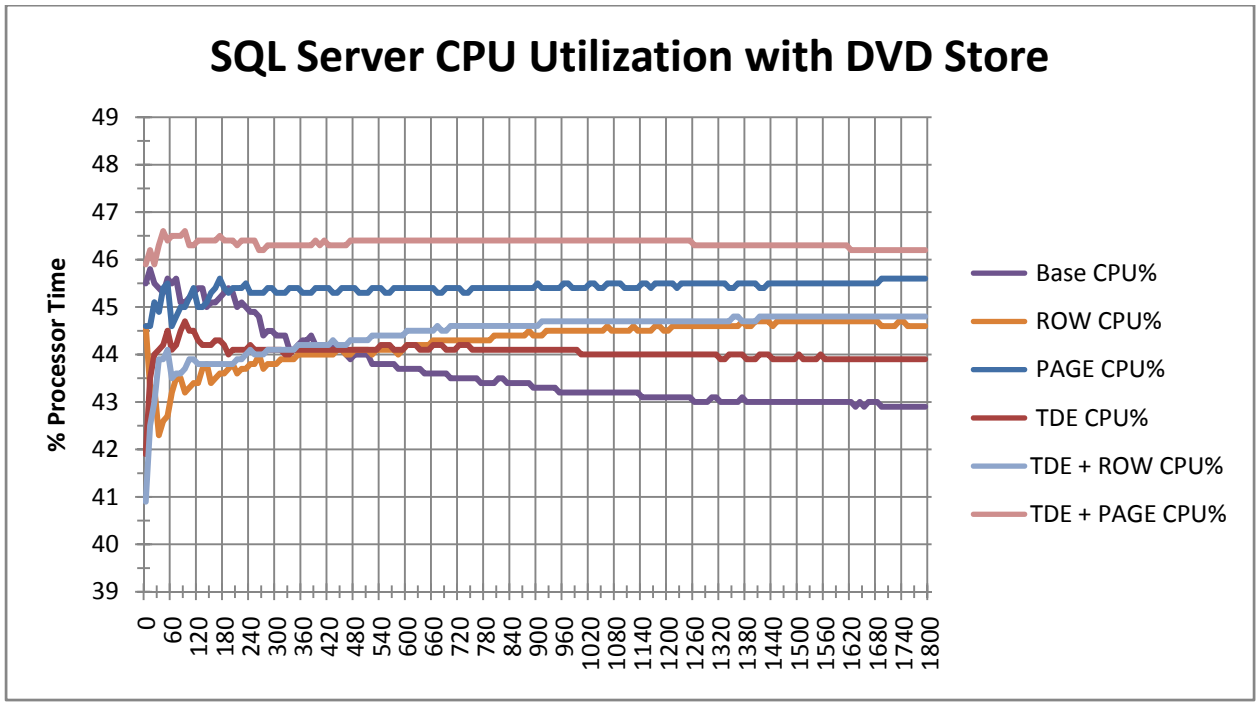

Figure 2: CPU Utilization with the Dell DVD Store Database

These observations show a nominal increase in processor utilization when encryption and compression were enabled. With this workload, compression was somewhat more costly than encryption, and—as expected—page compression required more resources than row compression. The total observed relative increase of 7.5% (with both TDE and page compression enabled) was lower than expected, and is likely attributable to the fact that the workload was not constrained by either the processor or I/O subsystems on the SQL Server host. For comparison, the same workload was also run against other server hardware. The absolute numbers were indicative of higher CPU utilization, but the relative increases were fairly consistent, varying by no more than 9% from the baseline when both page compression and encryption were enabled. If the workload included queries that required more processor resources, the impact of enabling compression and encryption is likely to be more noticeable.

Decision-Support Results

To simulate a decision-support system (DSS) workload, a set of car sales data was used. The database includes specific information about each vehicle that was sold, and tracks the sales activities of personnel at several dealers. The individual dealers are distributed across many states and designated sales regions. The data set includes historical information from several years of sales activities and is organized in a classic star schema, featuring a fact table and several associated dimensional tables. Because of the nature of these tests, the R710 SQL Server host was configured with two quad-core processors and 8 GB of RAM.

In order to simulate a DSS workload against the database, a series of stored procedures were created to allow an application to perform reports. The stored procedures performed full table scans against the fact table and joined against the dimension tables. Different sorting (e.g. ORDER BY) and filtering (e.g. WHERE, HAVING) options were employed, as would be expected for this type of workload.

For this set of tests, the data was not indexed. So, these stored procedures were able to simulate the type of ad-hoc data mining for which a DSS system is often employed. The procedures were run in batches so that multiple queries would execute in parallel, and the batches were submitted in a consistent manner against a set of six test configurations like those outlined in Table 2 (but with the relevant table and database names configured for this data set). Multiple iterations of each test configuration were run, and the results have been aggregated across these iterations.

Disk Subsystem Bandwidth and Throughput

For this type of database, the storage bandwidth is often one of the primary constraints. Throughput is less of a concern, because servicing DSS queries often produces large, sequential reads. As the workload was applied to each test configuration, we recorded both of these I/O criteria. The results of these tests are summarized in Table 4 and Table 5.

Table 4: Disk Bandwidth with the Car Sales Database

Configuration	Mean Read MB/s	Median Read MB/s	Maximum Read MB/s
Base	661.80	670.78	788.90
Row Compression	670.53	714.63	894.66
Page Compression	427.32	475.20	663.60
TDE	582.47	590.68	635.46
TDE + Row	489.11	498.33	560.78
TDE + Page	401.83	418.66	465.32

Table 5: Disk Throughput with the Car Sales Database

Configuration	Mean Reads/s	Median Reads/s	Maximum Reads/s
Base	1573.63	1544.60	3451.35
Row Compression	1448.32	1478.90	3026.65
Page Compression	929.08	980.54	2543.19
TDE	1932.20	1305.84	5397.32
TDE + Row	1141.74	1024.73	3776.99
TDE + Page	1742.74	1404.29	5451.77

Because the bandwidth was measured by the disk subsystem (which is not aware of the fact that data has been compressed), the MB/s values are expected to decrease when compression is enabled. The observations generally tended to meet this expectation. However, when Row compression was enabled, a negligible increase (~1.3%) was actually observed compared to the baseline configuration. This increase may be attributable to normal variances and error within the experimental design, but further testing would be required to confirm if that is the case.

The number of I/O operations required to retrieve the data also tended to decrease as compression was applied. The only test configuration where this trend failed to hold combined both encryption and Page compression. This configuration placed the host processor under a lot of stress, which likely contributed to this apparent anomaly.

Disk Space Used by the Database

The schema and design of a database play a key role in determining its compressibility. As was the case with the DVD Store, the car sales database employs the ***varchar*** type for most of its character data, and already implements smaller fixed types where appropriate. Since this data set was not indexed, only the space required to store the data is presented in Figure 3.

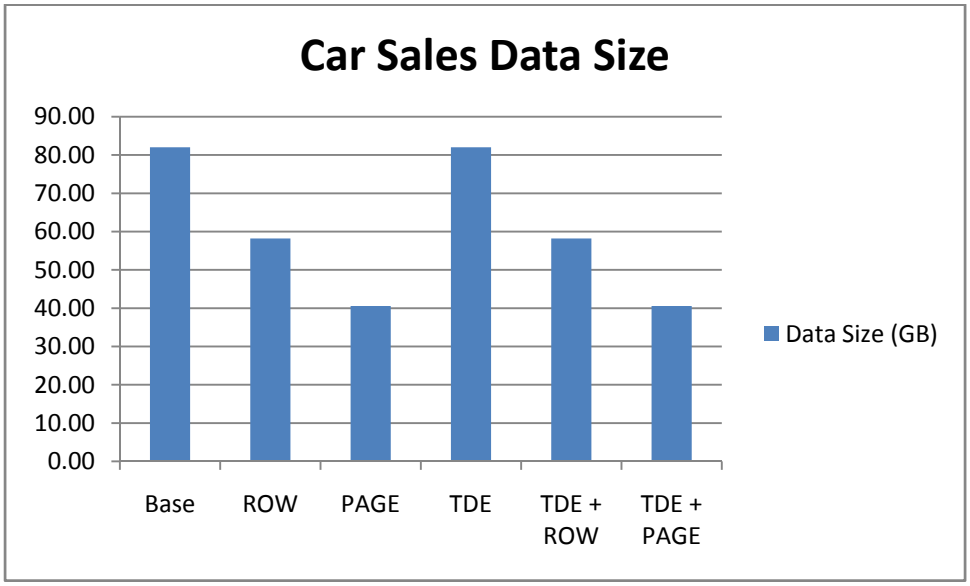

Figure 3: Disk Space Used by the Car Sales Database

For the car sales database, row compression offered a 29% savings and page compression offered a 50% savings compared to the uncompressed tables. These percentages are higher than those observed for the DS2 database, which is likely attributable to the fact that all of the car sales data was realistic, and therefore more compressible than the random data in the DVD Store data set. Encryption did not impact the size of the data; these reductions were realized whether TDE was enabled or not.

Processor Utilization

As the batched queries were executed, the CPU utilization was recorded. Because of the nature of the queries, there was a significant amount of variation during the execution of the workload. The average, median, and maximum values observed in each test configuration are captured in Table 6.

Table 6: CPU Utilization with the Car Sales Database

Configuration	Mean % Processor Time	Median % Processor Time	Maximum % Processor Time
Base	52.09	49.86	81.08
Row Compression	64.52	62.57	96.75
Page Compression	92.70	97.03	99.28
TDE	72.07	72.08	86.05
TDE + Row	77.89	77.43	95.45
TDE + Page	92.53	93.57	99.02

With this workload, there was a significant increase in processor resources when compression and encryption were enabled. This is attributable to the fact that the workload retrieves and processes large amounts of data in order to satisfy the queries. Without encryption, row compression saw a relative increase of 23%. Page compression saw a relative increase of nearly 77%, and consumed more resources than would typically be deemed acceptable.

Enabling TDE increased the CPU utilization by 38% compared to the baseline configuration. With encryption enabled, the relative increases in processor resource consumption were a further 8% for row compression and 28% for page compression. As with the unencrypted configuration, page compression pushed the limits of the processor subsystem. For the two test configurations that featured both encryption and compression, more resources were consumed than would generally be desirable in a production environment.

With the car sales data set, page compression was able to reduce the size of the data by 50%, but caused a large increase in processor utilization. This indicates that there could be an opportunity to balance investment between the storage subsystem and the server.

Aggregate Run Time

Unlike the DVD Store workload, which repeated a set of transactional tasks over the course of a fixed run-time, the car sales workload featured batched queries that were submitted in a repeatable fashion. The total run time in which all queries were successfully completed for each test configuration is recorded in Figure 4.

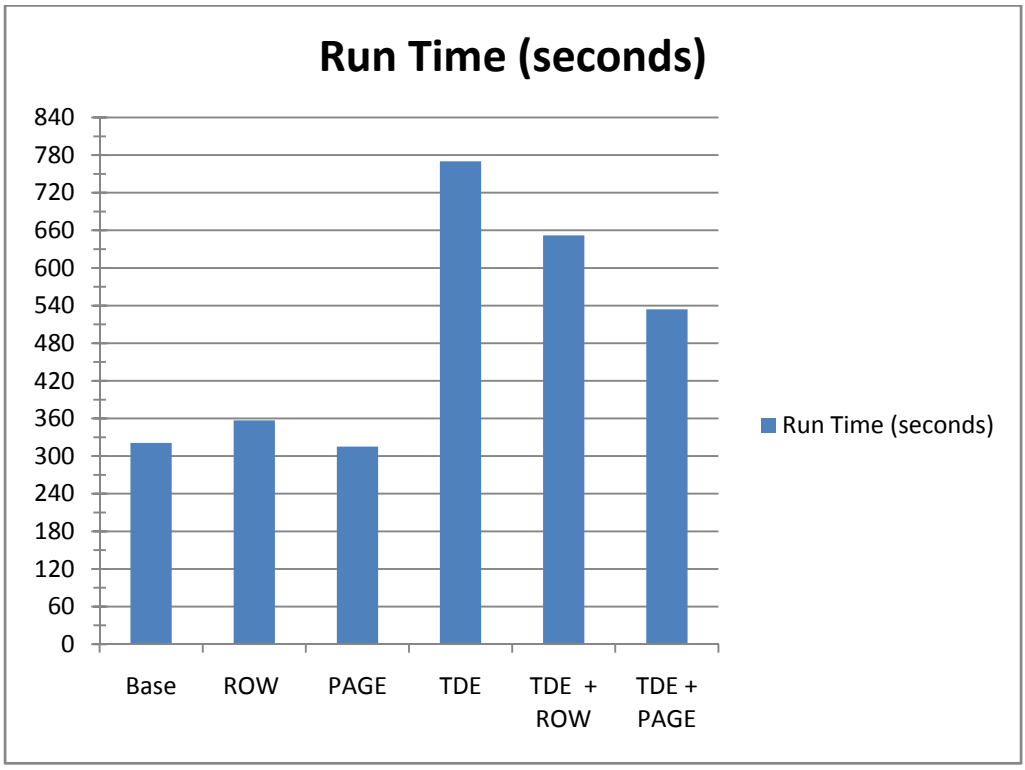

Figure 4: Time Required to Complete all DSS Queries

When compression is enabled, fewer I/O requests should be required to deliver the same amount of raw data, which is expected to reduce the time required to complete the queries. Compared to the baseline, the observed run time increased by 11% when row compression was enabled, but decreased by 2% with page compression enabled. This apparent anomaly is likely attributable to the observed increase in I/O during the row compression tests, which was accompanied by higher disk latencies. To validate the assumption, similar tests were conducted with a database schema similar to TPC-H. In those tests, the time required to execute a single complex query decreased as the level of compression was increased.

When encryption was enabled, the run-times were significant longer versus configurations without TDE. Combined with the increased CPU utilization, this change was likely caused by the fact that processor time needed to decrypt the data could not be used to process the incoming data. Test configurations with both TDE and compression required less time to complete the queries than the configuration that only featured encryption. As expected, the savings were larger with page compression than with row compression. To validate this result, additional tests were run with a database schema similar to TPC-H. In those tests, the time required to complete a single complex query increased by as much as 32% when encryption was enabled. However, the time savings associated with compression were still observed when TDE was enabled.

Conclusions

SQL Server 2008 Enterprise Edition offers two levels of data compression and provides a facility to encrypt data stored on disk. These features are all implemented via software, and are therefore expected to incur a cost in terms of CPU utilization. Encryption may be either necessary or desirable to help satisfy security or regulatory needs. Compression may reduce the amount of I/O that is needed to transfer data from disk to the SQL Server engine, which could provide benefits to some database workloads. It is important to understand how these features may impact a database host under a variety of different workloads. The Dell DVD Store (DS2) database was chosen to represent an OLTP workload, and a set of car sales data that was arranged in a data warehousing star schema was used to represent a DSS workload.

The conclusions below are drawn based on experiments that were conducted with these databases. In practice, every database will behave somewhat differently. Conducting experiments against representative sets of data by leveraging a development or test environment is strongly encouraged prior to implementing these features on a production database.

Compressing OLTP Workloads

The key concern for transaction processing workloads is often tied to having sufficient capacity to process the number of I/O operations per second demanded by the database. Historically, the most common way to increase the IOPS was to add additional physical disks to the database storage subsystem. The results of these experiments indicate that the data compression options native in SQL Server 2008 may provide an alternative in some cases. When data is compressed, fewer IOPS are required to satisfy the needs of the OLTP workload. As the level of compression is increased, these savings are amplified.

While our tests revealed almost-negligible increases in CPU utilization, this may not be the case with workloads that place more stress on the processor or disk subsystems. In such cases, the increase in processor utilization is expected to increase. Our results suggest that compression can provide an opportunity to balance the cost of an OLTP solution by making trade-offs between the storage and processing subsystems.

Encrypting OLTP Workloads

Many transaction processing databases store personally-identifiable information, such as addresses and credit card data. This means that security is often a key design concern for these databases and the applications that they serve. TDE provides a means to encrypt the data on disk, and is complimentary to other techniques (e.g. SSL or TLS) that secure data in transit between application servers and the SQL Server host. Since OLTP workloads are not usually CPU-bound, the SQL Server host should have adequate capacity to handle encryption tasks with little impact to the system.

There is, however, a trade-off associated with enabling encryption. Testing revealed a ten to twenty percent decrease in I/O throughput versus configurations that were not encrypted. If security standards, business controls, or regulatory compliance drive a requirement for encryption, this reduction may be considered a cost of satisfying the imperative.

Compressing DSS Workloads

Decision-support systems usually demand a disk system that can not only provide the capacity to house all of the data, but can also move data quickly enough to satisfy the needs of data mining and reporting systems. While indexing is useful for predictable operations, these systems also tend to handle ad hoc queries for which indexes may be more costly than beneficial. While compression was observed to incur a large cost in terms of processor

utilization, this is balanced by both the reduced I/O bandwidth and reduced time that is required for ad hoc queries to complete.

Since Page compression is a superset of Row compression, it is not surprising that the impact was more significant as the level of compression was increased. In our tests, enabling Page compression raised the CPU utilization beyond thresholds that would be considered acceptable in many environments. Options to accommodate these increased requirements can be achieved through higher bin speed processors or by scaling up the number of processors in the server. Despite this, reduced I/O requirements and reduced query run times were still observed. Again, this presents an opportunity to balance a SQL Server-based DSS system by making trade-offs between the disk and processing subsystems.

Encrypting DSS Workloads

DSS systems house sensitive business data, and may also contain personally-identifiable information. Therefore, regulatory compliance or business controls may demand that data is encrypted. In such cases, administrators should be aware that TDE will incur a penalty. This penalty is manifested by increased processor utilization, decreased I/O performance, and longer run times for ad hoc queries.

While none of these costs are desirable, test results indicate that the benefits associated with compression are still apparent when TDE is enabled. Mixed use-cases that feature both compression and encryption will require even more processor resources. This requirement is counterbalanced by the reduced I/O requirements and improved query run times that were observed versus the TDE-only test configuration, and provides an opportunity to amortize the cost of processing power with savings associated with the disk subsystem.

Tables

Figures

Appendix A: T-SQL Used to Compress DVD Store Data

The following script was used to compress the desired tables and indexes from the DS2 database. This example enables row compression—to enable page compression, change the syntax of each instance to read **DATA_COMPRESSION = PAGE**. To disable compression, change the syntax of each instance to read **DATA_COMPRESSION = NONE**. The script to compress the tables in the car sales database was similar, with the database and table names having been modified accordingly.

```
USE [DS2]

ALTER TABLE [dbo].[CUST_HIST] REBUILD PARTITION = ALL
WITH
(DATA_COMPRESSION = ROW)

ALTER TABLE [dbo].[CUSTOMERS] REBUILD PARTITION = ALL
WITH
(DATA_COMPRESSION = ROW)

ALTER TABLE [dbo].[INVENTORY] REBUILD PARTITION = ALL
WITH
(DATA_COMPRESSION = ROW)

ALTER TABLE [dbo].[ORDERLINES] REBUILD PARTITION = ALL
WITH
(DATA_COMPRESSION = ROW)

ALTER TABLE [dbo].[ORDERS] REBUILD PARTITION = ALL
WITH
(DATA_COMPRESSION = ROW)

ALTER TABLE [dbo].[PRODUCTS] REBUILD PARTITION = ALL
WITH
(DATA_COMPRESSION = ROW)

ALTER INDEX [IX_PROD_SPECIAL_CATEGORY_PRODID]
ON [dbo].[PRODUCTS] REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = ROW )
```

The following statement can be used to verify that the compression state is consistent with your expectations. Note that the WHERE clause will only return information about tables and indexes that have been compressed.

```
USE DS2
SELECT object_id, index_id, rows, data_compression,
data_compression_desc
FROM sys.partitions
WHERE data_compression != 0
```

Appendix B: T-SQL Used to Encrypt the DVD Store Database

We used this script to encrypt the DS2 database. It assumes that the keys and certificates that are required to enable TDE have not yet been created. The items listed in bold italics are listed as examples and should be customized for your environment. The script to encrypt the car sales database was similar, with the database name having been modified accordingly.

```
-- create master key
USE master
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'SQL_TDE_Master';
GO

-- create server certificate
CREATE CERTIFICATE Server_Cert_Name WITH SUBJECT = 'Server DEK Cert';
GO

-- create encryption key for the DS2 database
USE DS2
GO
CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES_128
    ENCRYPTION BY SERVER CERTIFICATE Server_Cert_Name
GO

-- backup certificate and private key
use master
BACKUP CERTIFICATE Server_Cert_Name TO FILE = 'PATH_TO_CERT_FILE'
   WITH PRIVATE KEY ( FILE = 'PATH_TO_KEY_FILE' ,
   ENCRYPTION BY PASSWORD = '1ComplexP@ssword' );
GO

-- enable TDE for the DS2 database
USE DS2
ALTER DATABASE DS2 SET ENCRYPTION ON;
GO
```

The following statement can be used to check the encryption state of all databases in a SQL Server instance. The ***encryption_state*** column will display **2** while background encryption is taking place, and **3** after encryption completes. When the state is **3**, the ***percent_complete*** value will revert to **–** and remain at **– zero**.

```
-- check encryption status
SELECT database_id, encryption_state, percent_complete
FROM sys.dm_database_encryption_keys
```