Microsoft SQL Server I/O Basics
Chapter 2

SQL Server 2000 SP4 and SQL Server 2005

Author: Bob Dorr (SQL Server Senior Escalation Engineer)

Reviewers: Mike Zwilling (SQL Server Development Manager)
Paul Randal (SQL Server Lead Program Manager)
Mirek Sztajno (SQL Server Program Manager)
Steve Schmidt (SQL Server Software Development Engineer)
Peter Byrne (SQL Server Software Development Engineer)
Eric Christensen (SQL Server Software Development Engineer)
Ryan Stonecipher (SQL Server Software Development Engineer)
George Reynya (SQL Server Software Development Engineer)
Kevin Farlee (SQL Server Program Manager)
Burzin Patel (SQL Server Program Manager)
Wei Xiao (SQL Server Technical Lead)
Kathy Lu (SQL Server Engine Test)
Bob Ward (SQL Server Senior Escalation Engineer)
Suresh Kandoth (SQL Server Escalation Engineer)

Published: June 2006

SUMMARY: Outlines the I/O features and enhancements in SQL Server 2000 Service Pack 4 and SQL Server 2005.

Copyright

Contents

Introduction

Microsoft® SQL Server™ 2005 continues to innovate and extend I/O performance and reliability. With new innovations come new terms, designs, and algorithms.

The HYPERLINK "http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlIObasics.mspx"SQL Server 2000 I/O Basics document is a prerequisite to the information contained in this document. Read it before you read this paper.

As the administrator of a SQL Server 2005 installation, you will find that visibility into the SQL Server I/O subsystem has been significantly increased. A primary focus of the SQL Server 2005 I/O design was overall stability of performance and data integrity as these relate to the outside influences of the I/O subsystem/path. The new and extended features provide both more visibility and more capabilities. You can maintain a SQL Server 2005 installation with high confidence in the system.

This white paper introduces new terms, discusses maintenance and configuration issues, new and improved error messages, and I/O enhancements.

After reading this document you will better understand SQL Server I/O needs and capabilities.

Terms

This section defines new terms that are used both in this paper and in documents that are referenced by this paper.

Subsystem / Path

The I/O subsystem or path includes those components that are used to support an I/O operation. These include, but are not limited to, the operating system, drivers, disk drives, controller cards, and caches.

Checksum

Checksum is an error-detection scheme that uses a formulated numeric value to represent a grouping of bits. The same formula is applied to the grouping to verify that the accompanying numeric value is the same. If it is not, the data is considered to be changed or damaged.

In other words, a formula is applied to a given set of bytes. The same formula can later be used to examine the data for damage.

As a simple example, take the following 2-byte string value of “AB” and apply a formula to it (Value = Position + ASCII value). Checksum is the sum of the values.

Character	POS	ASCII Value	Formula Value
A	1	65	67
B	2	66	68
		Checksum	**135**

Now assume that the string was stored on disk but when read back it was “AC”.

Character	POS	ASCII Value	Formula Value
A	1	65	67
C	2	67	69
		Checksum	**136**

Comparing the checksum values indicates that the values do not match and damage has occurred to the data.

Clearly this simple example would not handle a change of the string to “BA.” However, the routines that are used by SQL Server and Microsoft Exchange are more sophisticated and detect position reversal as a damaged data error condition.

Read Retry

When a read from stable media returns an error, the read operation is tried again. Under certain conditions, issuing the same read returns the correct data. This is a serious I/O subsystem problem that should be addressed immediately to avoid SQL Server and overall system stability problems and data loss. Obtain assistance from your I/O vendor if you encounter retries.

DBCC Page Auditing

DBCC page auditing is the process of executing common DBCC consistency checks as pages are read from disk.

Time of Last Access

The time of last access is a caching algorithm that enables cache entries to be ordered by their access times. SQL Server 2005 changed its lazy writer so that it returns data pages to the free list based on the time they were last accessed. This is different from the reference count design that was used in SQL Server 7.0 and 2000.

Idle Server

An idle server provides the ability, for certain SQL Server SKUs, to become silent; the idle server suspends the execution of core tasks such as checkpoint and lazy writer. The suspension takes place when no user requests are present and wakes when a new request arrives. This is similar to the suspend and resume operations performed by the operating system.

Copy-On-Write / Snapshots

Copy-on-write is the act of making a copy of the data before a

modification is complete. SQL Server copy-on-write actions are used to maintain snapshot databases in SQL Server 2005. Before a data page in the database can be modified, the unchanged image of the page is written (copied) to the snapshot database. The snapshot contains only the original images of data pages that have been modified after snapshot creation. A union of the unmodified database pages and the snapshot pages provides the point-in-time view of the data.

SQL Server 2005 introduces the snapshot database feature for point-in-time databases and online DBCC operations. For more information, see Database Snapshots in SQL Server 2005 Books Online.

Sparse Files

Sparse files provide a way to save disk space for files that contain large sections of data that is composed of zeros. An NTFS file marked as sparse only allocates disk clusters for the data explicitly written by the application. Ranges of the file that have not been written to are represented by non-allocated space. When data is read from allocated ranges, the data is returned as it was stored. Data read from non-allocated ranges is returned as zeros.

For information on sparse file behavior, see HYPERLINK "http://msdn2.microsoft.com/en-us/library/ms175823.aspx" Understanding Sparse File Sizes in Database Snapshots (HYPERLINK "http://msdn2.microsoft.com/en-us/library/ms175823.aspx" http://msdn2.microsoft.com/en-us/library/ms175823.aspx) and HYPERLINK "http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/sparse_files.asp" Sparse Files (HYPERLINK "http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/sparse_files.asp" http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/sparse_files.asp) on the Microsoft Developer Network (MSDN).

For example, you can create a sparse file of 100 GB on a 20-GB drive. The storage space that is required for the sparse file is only that of the actual bytes written to the file and not the maximum file size.

SQL Server 2005 uses sparse files for snapshot database files and online DBCC operations against files stored on NTFS volumes.

Streams

NTFS volumes enable data files to have one or more secondary storage streams. These streams are all part of the same file; if the file is copied, all streams are copied. Each stream works independently, enabling individual data storage attributes and sizing.

SQL Server 2005 online DBCC checks are based on the database snapshot technology by using transient, secondary file streams to store point-in-time images of data pages when the DBCC is in progress.

> **Note:** Many common utilities do not expose the secondary stream information. This includes size.

For more information, see: HYPERLINK "http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/file_streams.asp" File Streams on MSDN (HYPERLINK "http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/file_streams.asp" http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/fs/file_streams.asp**)**

Harden or Hardening

To *harden* is to require that data be stored on stable media during a specific event; the data is not just in a cache that could be lost if power was lost. For example, when checkpoint is finished, the dirty data pages have been written to stable media and are considered to be hardened. As soon as the hardened state is declared, activities such as truncating the inactive part of the transaction log can occur because the log records are no longer needed for crash recovery.

Memory Scribbler

The SQL Server support team uses the term *scribbler* to indicate an unexpected memory change for memory where the caller is not the true owner.

The following code example shows the setting of values in illegal array positions.

```
char gchArray[10];
gchArray[-1] = 'B';    -- Underflows the array bounds
gchArray[10] = 'R';    -- Overflows the array bounds
```

The first example puts the value 'B' in the memory location just in front of the array allocation and the second puts an 'R' in the memory location just after the array allocation. In both cases, they 'scribble' in memory that the array does not own.

Another example of a scribbler is using a stale pointer.

```
char * pchData = new char[10];
DELETE [] pchData;

memcpy(pchData, 0, sizeof(char) * 10)); -- pchData is
not valid
```

The memory controlled by the pointer was released with the `DELETE []` operation and is no longer owned by the current code line. The memory could be reused by another thread and the memcpy would damage memory it did not own, thereby scribbling on it.

Scribbling occurs because of bugs. Some of these are SQL Server bugs but many others involve other components. Be cautious of third-party extended stored procedures, COM objects, and Linked Servers that may be present in the SQL Server address space.

Maintenance and Configuration

This section outlines maintenance and configuration issues that should be fully understood before deploying SQL Server.

Pull-the-plug power outage testing

Power outage testing is a fundamental and critical part of the administrator's data safety and integrity requirements. Because the number of caching, defragmentation, and other products that enhance I/O performance has greatly increased, it is important to perform *pull-the-plug* power outage tests before production deployment. Performing safety checks of SQL Server and all other system components before

production deployment is very important. Many products successfully meet pull-the-plug requirements when they are configured correctly. Incorrect configurations lead to data loss.

Pull-the-plug testing should be done by using either the production environment or a replica of this environment. This includes third-party products that will be used in production.

The following is a list of pull-the-plug situations. *To guarantee equipment safety, make sure that the correct electrical standards are followed.*

Remove the power cord that supplies power to the mother board. Most tests implement this by using a switched outlet.

Remove the power cord that supplies power to storage devices. This may be the same as the motherboard or it may be separate for external storage devices.

Disconnect external devices, wait and then reconnect the data cabling.

As with all power outage testing, ensure sufficient delay during the outage tests.

We strongly encourage you to establish the appropriate pull-the-plug scenario testing for database log and data files in addition to backup files. Test actual power outages in a variety of scenarios to guarantee data integrity. Some pull-the-plug scenario examples follow.

Establish a wide recovery interval during these tests. While this is likely not the production configuration, it helps expose misconfigured I/O subsystem components. We suggest setting the **sp_configure** recovery interval to a high value, such as 5000.

Scenario	Basic Testing Steps
Transaction Safety: Commits	Insert a known set of data and commit the transaction(s). Pull the plug. Restart the server. Validate database(s) with DBCC CHECKDB. Validate all committed data exists.

Transaction Safety: Aborts	Insert a known set of data and commit the transaction(s). Create several active transactions that modify data and leave the transactions open. Issue a CHECKPOINT in each database. Pull the plug. Restart the server. Validate database(s) with DBCC CHECKDB. Validate all committed data exists and all uncommitted modifications where rolled back.

Backup Safety	Take a full database backup. Take a transaction log backup. Start a series of known data insert, update, and delete transactions. While data modification is occurring, continue to take a series of log backups. Immediately after a backup is finished, pull the plug. Restart the server. Validate databases(s) with DBCC CHECKDB. Validate the state of the data. Restore the backups and execute appropriate validation steps.

Note: Always do these tests with the checksum option enabled on the databases.

Restore backups on secondary systems frequently to make sure that your complete backup strategy is functioning correctly. That way you can recover from a failure condition.

For more information about I/O caching requirements and SQL Server, see the following white papers on MSDN.

HYPERLINK "http://support.microsoft.com/kb/917047/en-us" Microsoft SQL Server I/O subsystem requirements for the tempdb database (HYPERLINK "http://support.microsoft.com/kb/917047/en-us" http://support.microsoft.com/kb/917047/en-us)

HYPERLINK "http://support.microsoft.com/kb/917043/en-us" Key factors to consider when evaluating third-party file cache systems with SQL Server (HYPERLINK "http://support.microsoft.com/kb/917043/en-us" http://support.microsoft.com/kb/917043/en-us)

Atomic writes

The I/O subsystem must support atomic writes. Controllers and I/O subsystem components should not allow physical writes to occur before all data is safely transferred to stable media. If writes can start before the complete I/O request has been transferred, a power outage leads to a partial write and data is torn and damaged.

Defragmenting disk drives

Because physical data access is the most expensive part of an I/O request, defragmentation can provide performance gains for SQL Server and other applications. Positioning related data close to each

other reduces I/O operation requirements.

Various defragmentation utilities are available on the market today. Some utilities enable defragmentation on open files, whereas others require closed-file defragmentation or perform better when used under closed-file conditions. Additionally, some utilities have transactional capabilities, whereas others do not.

When you evaluate a defragmentation utility for use with SQL Server, the utility should provide transactional data capabilities. Use defragmentation utilities that provide the following transactional data capabilities.

The original sector should not be considered moved until the new sector has been successfully established and the data successfully copied.

The utility should protect against a system failure, such as a power outage, in a safe way that enables the files to remain logically and physically intact. To guarantee data integrity, a pull–the-plug test is highly recommended when a defragmentation utility is running on a SQL Server-based file.

The Write-Ahead Logging (WAL) protocol requires the prevention of sector re-writes to avoid data loss. The utility must maintain the physical integrity of the file as long as it does any data movement. In fact, it should work on sector boundaries in a transactional way to keep the SQL Server files correctly intact.

The utility should provide appropriate locking mechanisms to guarantee that the file retains a consistent image for any modifications. For example, the original sector cannot be modified when it is copied to a new location. Therefore, a defragmentation utility could lose the write if modifications are allowed.

Make sure that you understand any write-caching strategies that the utility uses. Caching by such a utility might involve a non-battery-backed cache and could violate WAL protocol requirements.

Open-file defragmenting raises several possible issues that closed-file defragmenting typically does not.

Open-file defragmenting affects performance. Defragmentation utilities may lock sections of the file, preventing SQL Server from completing a read or write operation. This can affect the concurrency of the server that is running SQL Server. Contact the defragmentation manufacturer to learn how files are locked and how this could affect SQL Server concurrency.

Open-file defragmenting can affect write caching and ordering. Open-file-based utilities require I/O path components; these components must not change the ordering or intended nature of the write operation. If the write-through or WAL protocol tenants are broken, database damage is likely to occur. The database and all associated files are considered to be a single entity. (This is covered in many

Knowledge Base articles, SQL Server Books Online, various white papers, and in HYPERLINK \l "_Remote_Mirroring" Remote Mirroring later in this paper.) All writes must retain the original write-ordering sequences and write-through capabilities.

We always recommend performing a full backup before you defragment those locations that contain SQL Server database and backup files.

Backup hardening

The hardening of backup streams has existed for all Microsoft versions of SQL Server, but the topic has not been formally documented. To harden a backup means that all data written to backup media has been successfully stored on stable media; sections of the backup are not just held in the cache. If hardening of the backup is not correctly achieved, the subsequent truncation of the database transaction log could be performed prematurely.

For an example related to hardening of the backup stream, consider a power outage. The sequence occurs in the following steps.

The backup stream I/Os have finished, but the tail of the backup stream data is held in a cache.

The SQL Server backup processing expects that the data was stored on stable media (hardened) and it continues processing, truncating the database transaction log.

A pull-the-plug event occurs so the tail of the backup stream is lost.

Upon restart, SQL Server crash recovery successfully recovers the database without error.

However, the backup stream is missing data because of the power outage. No specific error about this damage is produced.

The backup strategy is now damaged and the administrator does not know about the damage until a restore attempt of the backup sequence finds the damaged stream. All log backups following this kind of event are unusable.

> **Note:** SQL Server 2005 improves the restore verification capabilities and extends checksum capabilities to backups, but only a full restore can be used to fully validate the backup strategy. *It is not safe to assume that a successful* `RESTORE VERIFYONLY` *immediately following a backup operation indicates all the data is correctly stored on stable media.* A caching system could provide the data from cache and the backup stream could still be exposed to data loss. Always contact a cache manufacturer to understand the boundaries of the cache and its relationship to stable storage. Make sure that you have a way to read the backup stream data directly from stable media for verification.

The SQL Server backup process does not use `FILE_FLAG_WRITE_THROUGH`, as it does for the database data and log files; instead, it enables system-level I/O buffering. To guarantee data integrity, at the end of a backup operation SQL Server uses `FlushFileBuffers` to force all data written to the backup steams to stable media and then closes the file.

Notice that for a FlushFileBuffers API call, the NTFS file system issues a write for all dirty pages in the file cache and then invokes IRP_MJ_FLUSH_BUFFERS to make sure that all disk caches are

flushed. All third-party caching systems should provide similar semantics to guarantee that data is correctly stored on stable media.

`FlushFileBuffers` is the mechanism that is used to harden the backup stream. Therefore, make sure that `FlushFileBuffers` achieves stable media storage because when the backup successfully completes the `FlushFileBuffers`, it closes the streams and then truncates the database transaction log.

One other caveat to remember about backup hardening involves moving or compressing the backup. Many backup strategies include the ability to take a backup and then compress it or copy/move it. The compression or copy/move operation is an extension of the implied WAL contract. In-place compression must include transactional capabilities (as outlined in HYPERLINK \l "_Defragmenting_Disk_Drives"Defragmenting Disk Drives earlier in this paper) or else the compression action exposes data loss possibilities. Always make sure that the integrity of the backup stream is maintained and correctly secured on stable media before destroying the original file.

4-KB disk sector sizes

New manufacturing techniques and specifications will produce products that include drives that support larger sectors than the current 512-byte sector formats where one 8-KB SQL Server data page requires sixteen 512-byte sectors. Alongside the increased sector size are various manufacturer implementations of the 512-bytes to 4-KB sectors. For example, actual sector sizes might be 512-bytes, 1-KB, and 4-KB. The increased sector sizes and manufacturer implementation derivations affect the SQL Server Write-Ahead-Logging protocol in several ways, which are described in this section.

Sector rewrite

Some of the current manufacturer implementations request a logical view of the on-disk sectors. This means the operating system sees a sector size of 512 bytes but the drive performs the appropriate logical-to-physical mapping to the larger sector sizes as stored on physical disk. A logical sector write can include the reading of nearby data known as Read Modify Write (RWM) to finish the complete sector-sized write.

For more information on logical to physical sector mapping, see HYPERLINK "http://www.t13.org/docs2005/e05122r3-WD_Comments_on_Long_Sectors.pdf" Implementation Guidelines for 1K/4K Sector Sizes (HYPERLINK "http://www.t13.org/docs2005/e05122r3-WD_Comments_on_Long_Sectors.pdf" http://www.t13.org/docs2005/e05122r3-WD_Comments_on_Long_Sectors.pdf).

The SQL Server transaction log files are always written with sector-aligned sizes and at sector-aligned boundaries. A small-logical to larger-physical mapping action causes the rewrite (RMW) of log sectors that were already saved to stable media. If the subsequent write fails,

the transaction log is damaged and previously committed transactions can be lost. This is termed a tear or torn condition of the already hardened log.

> "… SQL Server 6.*x* may not see the same performance impact from frequent and small transaction log writes. SQL Server 6.*x* rewrites the same 2-KB log page as transactions are committed. This can reduce the size of the log significantly compared to the 512-byte sector boundary flushes in SQL Server 7.0, SQL Server 2000, and SQL Server 2005. Reducing the size of the log directly relates to the amount of mechanical drive activity. However, as explained above, the SQL Server 6.*x* algorithm may expose committed transactions…."

For this and more information on how SQL Server 7.0 changed the transaction log design to prevent sector rewrites, see HYPERLINK "http://support.microsoft.com/kb/230785/en-us" SQL Server 7.0, SQL Server 2000, and SQL Server 2005 logging and data storage algorithms extend data reliability on MSDN (HYPERLINK "http://support.microsoft.com/kb/230785/en-us" http://support.microsoft.com/kb/230785/en-us).

Logical-to-physical mapping implementations may be dangerous and could lead to complex data loss situations.

Align with physical sector boundary

Implementations that present logical sector sizes introduce various complexities for SQL Server and the operating system. For this discussion, we use an actual sector size of 4 KB and a presented (logical) sector size of 512 bytes.

SQL Server database log or data files should be created on a physical sector-aligned boundary. However, because the operating system is presented with sectors of size 512 bytes, it can align the start of the SQL Server file at a location that is not aligned with the physical sectors.

The following table shows how the logical sector presentation can cause SQL Server database log and data files to span sectors in an unaligned way. Sector read, modify, writes (RMWs) occur in order to handle the leading and trailing data boundaries.

As shown in the table, a write to Database File Page 0 must handle sectors from File ABC and File XYZ, which is at the head of the I/O in addition to Database File Page 1, which is at the tail of the I/O. I/O performance can decrease because of the extra misaligned activities.

Physical 4KB Sectors	Logical 512-Byte Sectors

Sector #1	File ABC File XYZ Database File Page 0 – Sector 1 of 16 Database File Page 0 – Sector 2 of 16 Database File Page 0 – Sector 3 of 16 Database File Page 0 – Sector 4 of 16 Database File Page 0 – Sector 5 of 16 Database File Page 0 – Sector 6 of 16

Sector #2	Database File Page 0 – Sector 7 of 16 Database File Page 0 – Sector 8 of 16 Database File Page 0 – Sector 9 of 16 Database File Page 0 – Sector 10 of 16 Database File Page 0 – Sector 11 of 16 Database File Page 0 – Sector 12 of 16 Database File Page 0 – Sector 13 of 16 Database File Page 0 – Sector 14 of 16
Sector #3	Database File Page 0 – Sector 15 of 16 Database File Page 0 – Sector 16 of 16 Database File Page 1 – Sector 1 of 16 Database File Page 1 – Sector 2 of 16 Database File Page 1 – Sector 3 of 16 Database File Page 1 – Sector 4 of 16 Database File Page 1 – Sector 5 of 16 Database File Page 1 – Sector 6 of 16

You might be tempted to work around this by creating a database on a freshly formatted drive so that an alignment issue would not occur. The first fault with this workaround is that there is no guarantee of physical sector alignment when the operating system is presented a logical sector size. Also, it is generally impractical on a production system.

Another clever workaround would be to use a defragmentation utility. The fault here is that the defragmentation utility is likely to work from

the logical sector size and not the physical sector size. Although better data alignment proximity might occur, it is just as likely that the movement using logical sector boundaries could lead to sector-spanning situations shown in the example.

Nevertheless, another workaround would be to back up and restore the database to achieve better physical alignment. To achieve new physical allocations, drop the original database and run the restore. The new file creation and data copy may align with the physical sectors better or even reduce logical fragmentation. However, the start of the file could be unaligned and the restore attempt could cause all database pages to be unaligned as in the example shown earlier.

The peak of poor logical-to-physical fragmentation could be as bad as `Max Sector Frags = Physical Sector Size / Logical Sector Size`. In the example, this could create a `16:1` ratio of sectors to SQL Server 8-KB data page space, causing pages of the database to be stored on 16 unique sectors, intermixed with other data.

SQL Server lets you grow and shrink database log and data files. This causes the acquisition and release of disk space. This can lead to further logical-to-physical fragmentation.

An example that uses files that can automatically grow is helpful. In this example, the original database was 10 GB with an auto-grow increment of 1 GB. The original 10-GB allocation occurred on a clean drive and enabled all data pages to align on the physical sector size correctly. However, the same drive supports storage of backups (this is a poor practice) and other files. Data is added to the database that causes an auto-grow but the growth acquires the 1-GB portion from the drive starting at an unaligned physical sector location. This leaves part of the database aligned and part of it unaligned to physical sector boundaries.

The performance implications may be such that data values stored in the original 10-GB pages can be read and written faster than that in the new 1-GB section of the database. This would be a very difficult issue to track down and correct.

Align with 8-KB boundary

Sector sizes should always allow for 8-KB alignment of SQL Server data pages. SQL Server writes data pages in 8-KB increments; the physical sector sizes and alignment should always enable SQL Server to write the 8 KB so that it is aligned on sector boundaries or as a multiple of a smaller sector size. This is to prevent torn I/O conditions, data page spanning, and sector rewrites.

Sector sizes of 4 KB enable the SQL Server data page to occupy two physical sectors and allows for SQL Server data pages to be aligned on 8-KB boundaries efficiently.

If a sector size of 1536 bytes is used, the alignment for data pages is

broken. This creates 8-KB pages that span the same physical sector. This results in the risk of unwanted sector rewrites (RMWs) of data pages. This hinders database performance and could lead to unexpected data loss (torn) situations.

Larger transaction logs

The SQL Server transaction log files are always written with sector-aligned sizes and at sector-aligned boundaries. Most ordinary workloads do not require SQL Server to significantly increase log space usage due to larger physical sector sizes. For example, concurrent transactions share log-block space.

Although SQL Server tries to use the log space as efficiently as possible, certain application patterns cause the log-block fill percentages to remain small. The following extreme example points out the clear difference in the transaction log space requirements as the sector size varies.

```
  WHILE(@I < 10000)
  BEGIN
    BEGIN TRAN
    INSERT INTO tblTest values ('A', @I)
    COMMIT TRAN
    SET @I = @I + 1
  END
- vs -
   BEGIN TRAN
  WHILE(@I < 10000)
  BEGIN
    INSERT INTO tblTest values ('A', @I)
    SET @I = @I + 1
  END
  COMMIT TRAN
```

The first example requires SQL Server to write the log records 10,000 times, as each commit is processed. The second example enables SQL Server to pack log records and write all 10,000 inserts and the single commit at the same time.

Calculating the log usage for the examples results in the following approximations.

By using a 4-KB sector size for the first example, ~40 MB of disk space is used.

By using a 512-byte sector size for the first example, ~5 MB of disk space is used.

By using a 4-KB sector size for the second example, ~1 MB of disk space is used.

By using a 512-byte sector size for the second example, ~1 MB of disk

space is used.

This means for an I/O subsystem that reports sector sizes larger than 512 bytes, the SQL Server transaction log file could acquire physical space at the rate of "Rate = n / 512" where "n" is the new sector size. Application design can become critical to efficient log space usage.

> **Note:** Be cautious. Expanding the length of a transaction can have adverse affects on concurrency as locks continue to be held until the commit is finished.

Although **tempdb** is primarily non-logged, internal operations such as the allocation of a data page can be logged. Therefore, larger sector sizes can affect **tempdb** size.

Restore and attach

A SQL Server database can be restored or attached on a system that has a smaller sector size. To help guarantee data integrity, the larger sector size should be evenly divisible by the smaller sector size. For example a 4-KB source restored/attached to a 1-KB or 512-byte destination is an evenly divisible size. Restoring or attaching to a smaller sector size such as 1536 bytes does not fill the "evenly divisible" requirement and immediately requires sector rewrite operations to occur.

SQL Server is not designed to dynamically upgrade the database to the larger sector sizes. SQL Server disallows restoring or attaching a database on a system with a larger sector size; it generates an error message and prevents the restore or attach operation. Enabling a database to function by using a smaller formatted sector size than the actual sector size violates the WAL protocol because the sector size variation guarantees the log records will not be correctly aligned with the physical sector size and log records will be rewritten.

At the time of publication, some current subsystems report sector sizes larger than 512 bytes but most do not. The larger sector sizes involve newer technology and require API changes at the operating system level. Future changes in Microsoft® Windows and SQL Server will correctly support the larger sector sizes and allow for dynamic adjustment of sector sizes.

Format for 4-KB sectors run on smaller sectors

SQL Server prevents restoring or attaching a database in an environment that has a larger physical sector size than the sector size the database was formatted with.

SQL Server can restore or attach a database formatted with a larger sector size onto smaller, evenly divisible physical sectors. This is possible because the log write uses the original formatted size (the larger size of the two). SQL Server may use a bit more log space in this configuration, but the writes to the log are larger than the physical sector size. This prevents rewrites as long as the smaller sector size is

an even divisor. An irregular division of (original sector size / physical sector size) creates rewrites (RMWs) and should not be allowed.

System and sample databases

SQL Server 2005 ships all system (**master**, **model**, and **msdb**) and sample databases formatted with 4-KB sector sizes so that they can be installed on a device with up to 4 KB. SQL Server 2000 system and sample databases are 512-byte based, causing Setup to fail on larger-sector drives.

The model database is used as the template when SQL Server creates **tempdb** and user databases. Only the contents of the **model** database are used, not the full physical format. **tempdb** and new database creations use the sector size reported by the operating system at the time of creation. This is performed on a per-file basis. Variance of sector sizes can then occur across the files throughout a database. Make sure that the sector sizes of a given I/O path are equal.

Determining the formatted sector size of database

Run the `DBCC fileheader('<<dbname>>')` command to output the formatted sector size for each file in the database. The `SectorSize` column shows the formatted sector size in bytes.

What sector sizes does SQL Server support?

SQL Server currently supports the following sector sizes that are equal to or less than 4 KB.

Physical sector sizes that evenly divide into 4 KB

Physical sector sizes that are smaller than that of the database's original formatted sector size as long as the smaller sector size is evenly divisible into the originally formatted sector size

512

1024

2048

4096

Remote mirroring

Several hardware vendors provide remote mirroring solutions. Remote mirroring captures writes and duplicates them remotely. Because of the distance between the principal and mirror and the reluctance to affect the performance on the principal by making it wait for remote writes, remote mirroring systems generally operate asynchronously. Very strict rules and protocols must be adhered to in order to safely use a remote mirroring technology with SQL Server data, log, and backup files.

For more information on SQL Server support for remote mirroring solutions, search MSDN for the Using Third Party Mirroring Solutions with SQL Server 2000 and 2005 User Databases white paper.

The following is a brief recap of those attributes beyond the local WAL protocol that are necessary for a successful remote mirroring deployment.

Write ordering: The order of the writes on the primary must be the same exact order as that used to apply changes to the mirror. If the write order is not maintained, the WAL protocol is destroyed and the database will become corrupted.

Consistency groups: A consistency group makes all I/O actions to a set of volumes appear as a single I/O stream, keeping all write requests in exact order across the group. Without the ability to keep write ordering across a group of volumes intact, it is not safe to mirror SQL Server databases that span volumes. If each volume can keep a separate write order, the database log and data files on the mirror will not maintain the correct point-in-time semantics and the database will become corrupted.

Notice that consistency groups can apply to local mirror solutions as well.

Restore points: Following is an example of the *restore points* or *forks* in a remote mirroring system.

Primary	**Mirror**
Full Database Backup	
Log Backup #1 (LSN 1 to 100)	
Log Backup #2 (LSN 101 to 200)	
FAILOVER OCCURS	
	Log Backup #3 (LSN 101 to 150)
	Log Backup #4 (LSN 150 to 200)

The restore point issue occurs at LSN 101. If the remote mirroring solution does not guarantee immediate delivery to the mirror, some write operations can be lost under a failure condition. In the example, the LSN changes from 150 to 200 at the primary are assumed to have been 'in-play' at the time of the failover and never made it to the mirror. The mirror is brought online and therefore loses the data between 150 and 200 because the data never made it to the mirror. The database is transactionally consistent on the mirror after recovery but some data loss has occurred.

The log backups start after failover as shown in the example. At this point there are two outstanding issues which must be correctly resolved.

The primary may have changes on disk that never made it to the mirror and therefore requires a full synchronization with the mirror. Your manufacturer may provide a more direct way to resynchronize the primary after failover but until the primary is in synchronization with the mirror, it is not safe to use the original primary again.

During restore, the primary 'Log Backup #2' must be considered invalid. The restore sequence would be the following:

Restore the Full Database Backup.

Restore Log Backup #1.

Restore Log Backup #3. If Log Backup #2 is restored, log backups #3 and #4 cannot be restored because log records 150 to 201 on the mirror differ from originally lost on the primary.

Restore Log Backup #4.

Microsoft SQL Server 2005 I/O Error Message Changes and Additions

SQL Server 2005 has more error and message context information than did previous versions. This section outlines the significant I/O error message changes and additions.

Error 823

Error message 823 has been split into different error messages in order to provide improved context. Error message 823 in SQL Server 2005 represents an I/O transfer problem and error message 824 represents logical consistency problems. The 823 error message indicates a serious system error condition requiring the operating system issue to be resolved in order to correct the problem.

The message example shown here is the improved 823 error message text.

> The operating system returned error `<<OS ERROR>>` to SQL Server during a `<<Read/Write>>` at offset `<<PHYSICAL OFFSET>>` in file `<<FILE NAME>>`. Additional messages in the SQL Server error log and system event log may provide more detail. This is a severe system-level error condition that threatens database integrity and must be corrected immediately. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online.

SQL Server error 823, occurs when any one of the following API calls give you an operating system error.

ReadFile

WriteFile

ReadFileScatter

WriteFileGather

GetOverlappedResult

For extended details on the 823 error, see HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;828339" Error message 823 may indicate hardware problems or system problems (HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;828339" http://support.microsoft.com/default.aspx?scid=kb;en-us;828339) on the Microsoft Web site.

During read operations, SQL Server 2005 may perform read retries before recording that an 823 error condition has occurred. For details, see HYPERLINK \l "_Read_Retry" Read Retry later in this paper.

Error 824

The 824 error indicates that a logical consistency error was detected during a read. A logical consistency error is a clear indication of actual damage and frequently indicates data corruption caused by a faulty I/O subsystem component.

The example text of the 824 message is shown here.

> SQL Server detected a logical consistency-based I/O error: <<ERROR TYPE DESCRIPTION>>. It occurred during a `<<Read/Write>>` of page `<<PAGEID>>` in database ID `<<DBID>>` at offset `<<PHYSICAL OFFSET>>` in file `<<FILE NAME>>`. Additional messages in the SQL Server error log or system event log may provide more detail. This is a severe error condition that threatens database integrity and must be corrected immediately. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online.

Error types

The 824 message contains extended details about each specific logical error as outlined in the following table.

> **Note:** An 824 error indicates a serious I/O subsystem stability problem and should be corrected immediately.

Error Type	Description
Checksum	The read resulted in a checksum failure. The checksum stored on the data page does not match the checksum as calculated after the read operation. Data on the page has been damaged and will require a restore to correct it. Extended Data: "incorrect checksum (expected: ##; actual: ##)" Contact your hardware manufacture for assistance.
Torn Page	The read resulted in a torn bits failure. The torn bits stored in the data page header do not match the torn bits stored in the individual sectors following the read operation. Data on the page has been damaged and will require a restore to correct it. Extended Data: "torn page (expected signature: ##; actual signature: ##)" Contact your hardware manufacture for assistance.
Short Transfer	The requested number of bytes were not read. For example, if the read request was for 8 KB but the returned data was only 4 KB, the condition is flagged as a short transfer error. This indicates that the file is damaged or the I/O subsystem has a severe problem transferring data to and from media. Extended Data: "insufficient bytes transferred"

Bad Page Id	The page header does not contain the correct value for the expected page ID member. The expected page ID can be calculated using the following formula: (`page id = physical offset in file / 8192 bytes`). When the expected page is not returned, the bad page ID error is indicated. Extended Data: "incorrect pageid (expected ##:##; actual ##:##)" This is frequently a condition where the I/O subsystem returns the incorrect data during the read. Microsoft SQL Server Support investigations of these typically reveal that the I/O subsystem is returning data from the wrong offset in the file or the page contains all zeros. Contact your hardware manufacture for assistance.
Restore Pending	By using SQL Server 2005 Enterprise Edition, a single page restore can be performed to correct a corrupt page. If a page is damaged, it is marked as a bad page and any attempt to access it returns an 824 error. This indicates that a restore is required to correct the damaged page before it can be accessed. Extended Data: "Database ID `<<DBID>>`, Page `<<PAGEID>>` is marked RestorePending. This may indicate disk corruption. To recover from this state, perform a restore."

Stale ReadFor details about stale read errors, see HYPERLINK \l "_Stale_Read_Protection" Stale Read Protection later in this paper. The behavior is controlled with trace flag –T818.

Briefly, if a page has been recently written to disk and is still stored in the stale read hash table, the Log Sequence Number (LSN) stored in the hash table is compared to the LSN in the page header. If they do not match then the page is flagged as incorrect. Example message: "stale page (a page read returned a log sequence number (LSN) (##:##:##) that is older than the last one that was written (##:##:##))"	
Page Audit Failure	When trace flag `-T806` is enabled, a DBCC audit is performed on the page to test for logical consistency problems. If the audit fails, the read is considered to have experienced an error. Extended Data: "audit failure (a page read from disk failed to pass basic integrity checks)" Page auditing can affect performance and should only be used in systems where data stability is in question.

Error 832

Error message 832 is returned when the in-memory checksum audit fails. For details about the in-memory checksum design, see HYPERLINK \l "_Checksum" Checksum in the Microsoft SQL Server 2005 Enhancements section in this document.

Following is an example of the text of the 832 error.

> A page that should have been constant has changed (expected checksum: `<<VALUE>>`, actual checksum: `<<VALUE>>`, database `<<DBID>>`, file `<<FILE>>`, page `<<PAGE>>`). This usually indicates a memory failure or other hardware or OS corruption.

The 832 message indicates a serious process stability problem, such as a scribbler, that could lead to data corruption and loss.

Error 833

SQL Server 2000 SP4 and SQL Server 2005 include stalled I/O warnings as described later in this document. The following is an example of the 833 text which is written in the SQL Server error log.

> SQL Server has encountered `<<##>>` occurrence(s) of I/O requests taking longer than `<<##>>` seconds to complete on file [`<<FILE>>`] in database [`<<DB NAME>>`] (`<<DBID>>`). The OS file handle is `<<HANDLE>>`. The offset of the latest long I/O is: `<<PHYSICAL OFFSET>>`

The 833 message indicates an I/O is hung, or is just taking a long time. This is likely an I/O subsystem problem. The information in the message can be used by Microsoft Platforms Support or your I/O subsystem vendor to trace the specific IRP and determine the root cause.

The following are a few reasons this error may be encountered.

Malfunctioning virus protection

Heavy use of compression

Network unresponsiveness

Dual I/O path software malfunctions

Microsoft SQL Server 2005 Enhancements

The following section outlines the core I/O enhancements made in SQL Server 2005.

Checksum

SQL Server 2005 introduces the ability to checksum data pages, log blocks, and backups. For details on checksum capabilities and usage, see the ALTER DATABASE topic in the PAGE_VERIFY section in SQL Server 2005 Books Online.

The expansion of hardware capabilities along with the increased use of virus protection, caching mechanisms, and other advanced filter drivers increase the complexity of the I/O subsystem and expand the point-of-failure possibilities. Microsoft SQL Server 2005 and Microsoft Exchange Server products provide checksum capabilities to enhance data protection.

The checksum algorithm used by SQL Server 2005 is the same algorithm used by Microsoft Exchange Server. The SQL Server algorithm has an additional rotation to detect sector swaps.

Microsoft Exchange Server introduced checksum capabilities several years ago with great success. Search the Microsoft Knowledge Base for more information about error message -1018, which indicates a checksum failure for the Exchange Server product. The following is an excerpt from the Exchange Server Knowledge Base article HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;151789" KB151789.

> "When you perform a transaction with the Jet database, the information store or the directory store writes the transaction to a transaction log file (Edb*.log in Mdbdata or Dsadata). The transaction is then committed to the Jet database. During this process, the Jet engine calculates the page's checksum value to be written, records it in the page header, and then requests that the file system writes the 4-KB page of data to the database on disk.
>
> …
>
> Even after you restore from a known good backup, however, the `-1018` errors may appear again unless the root causes of the physical data write problems are resolved."

The checksum algorithm is not an ECC or CRC32 implementation but a much less CPU-intensive calculation that avoids affecting database throughput.

The data page and log throughput affects are limited by the buffer pool caching and read-ahead designs. This enables the writes and reads to be done out-of-critical-band when it is possible.

Writes

SQL Server data pages are typically written to disk by the checkpoint or lazy writer processing.

SQL Server determines when to run checkpoint activity based on the `sp_configure 'recovery interval'` goal and the amount of log space currently being used.

SQL Server 2005 determines when to write dirty pages from the buffer pool cache based on memory pressure and time of last access of the page.

Checksums are calculated immediately before the data page or log block is written to disk. SQL Server tries to perform writes in groups and in a background manner whenever possible to avoid directly affecting user queries. The caching of data pages and grouping of log records helps remove much, if not all, of the command latency associated with a write operation. As described, the checksum calculation activity can frequently be done out-of-band from the original request, thereby reducing any direct affect checksum may add to the write.

> **Note**: The model database is checksum (page audit) enabled. Therefore, all new databases created in SQL Server 2005 are checksum enabled to maximize data protection.

Reads

When a page or log block is read from disk, the checksum (page audit) value is calculated and compared to the checksum value that was stored on the page or log block. If the values do not match, the data is considered to be damaged and an error message is generated.

SQL Server uses read-ahead logic to avoid query stalls caused by I/O waits. The read-ahead design tries to keep the physical reads and checksum comparisons out of the critical path of the active query, decreasing the performance effects of checksum activity.

Damage

The checksum is designed to detect whether one or more bits of the data unexpectedly changed; it was not designed to correct problems.

The checksum is calculated immediately before the write to disk and verified immediately after the physical read from disk. If damage is detected, this indicates a serious I/O subsystem data integrity problem and the I/O subsystem should be thoroughly checked for problems. A failure indicates that data being written to and retrieved from stable media did not maintain its integrity.

Disk drives, caches, filter drivers, memory, CPUs, and other components should be reviewed in complete detail if the system

reports checksum failures. Be cautious of power outages as well.

PAGE_VERIFY usage

The `ALTER DATABASE` command is used to change the database's `PAGE_VERIFY` protection settings. There are three possible settings; `NONE`, `CHECKSUM`, and `TORN_PAGE_DETECTION`. The database maintains the verification setting. A status value in each page header indicates the type of protection and verification values stored when data was written to stable media.

Similar checksumming activity occurs for log block writes and reads when CHECKSUM protection is enabled. Log writes and reads always use a parity bit design (torn protection) to mark the valid blocks in the log. An additional checksum of the log block is new and is applied only when the database checksum verification option is enabled.

The following table outlines the verification actions SQL Server 2005 performs based on the database's `PAGE_VERIFY` option and the page's status value, which is located in the page header. Some of the actions in this table might not seem correct because the page's status value on a read appears to override the database's current setting. However, on a read the possible verify action is determined from the page header status and not from the current database setting.

For example, a checksum cannot be checked on the read if the checksum wasn't calculated and stored during the write of the page.

Page Header Setting	Actions Before Write	Actions After Read

NONE	The status of the page header is set to NONE for page verify protection. This maximizes performance but provides NO physical integrity protection beyond that provided by the I/O subsystem itself. This is not a recommended setting and should be used with caution. Backup plans are especially important for databases that are set to the page verify option of NONE.	The page was not stored with any protection values, so no verification occurs during a read. **Page Header Status = NONE** **Database's Page_Verify Setting** **Protection Check** NONE NONE TORN NONE CHECKSUM NONE

CHECKSUM	The checksum formula is applied to the 8-KB data page. The page header checksum value is updated and the page header status is set to CHECKSUM. The page is then written to stable media. Checksum protection uses the most CPU cycles of the three options because it must examine all bits on the page. However, the algorithm has been tuned and the resulting affect is minimal. Checksum is the default database setting in SQL Server 2005.	If a page is read that was written with either checksum or torn page protection, verification occurs for the type of protection indicated in the page header. **Page Header Status = CHECKSUM** **Database's Page_Verify Setting** **Protection Check** NONE NONE TORN CHECKSUM CHECKSUM CHECKSUM

TORN	The TORN page protection is established by writing a 2-bit value in the lowest order 2 bits of each 512-byte sector of the page. The page header is updated with the torn bit tracking information and the page header's verify status is set to TORN. The page is then written to disk. Because the TORN protection uses only 2 bits in each sector of the 8-KB page, it requires less CPU cycles but provides far less protection than checksum.	If a page is read that was written with either checksum or torn page protection, verification occurs for the type of protection indicated in the page header. **Page Header Status = TORN** **Database's Page_Verify Setting** **Protection Check** NONE NONE TORN TORN CHECKSUM TORN

Note: SQL Server does not rewrite all database pages in response to an `ALTER DATABASE PAGE_VERIFY` change. The `PAGE_VERIFY` option can be changed over time, and pages written to disk will reflect the option that was in effect at the time they were written. Therefore, the database can have pages in any one of the three available verification states.

There is no single command that establishes a `PAGE_VERIFY` option and applies it to all pages of the database. This includes backup and restore.

Backup and restore operations maintain the same physical data integrity as the original database. Backup and restore operations do provide a checksum option but this is different from the `PAGE_VERIFY` option.

Rebuilding clustered indexes in the database can dirty most of the data and index pages and achieve broad page protection establishment. However, heaps, text/image, stored procedures, stored assemblies, and others are not dirtied by a clustered index rebuild operation.

Only reuse of the transaction log blocks with the appropriate protection can apply the specified protection to the log blocks.

The only way to make sure that all user data pages contain the desired page verification protection is to copy all data, at the row level, to a new database that was created by using the appropriate page verification option.

In-memory checksums

SQL Server 2005 extends the protection of data pages by extending the PAGE_VERIFY CHECKSUM to allow for in-memory checksumming. There are limited situations for which this is helpful, such as in-memory scribblers, uncertainty about page file stability, and uncertainty about RAM memory stability.

The same checksum algorithm used by `PAGE_VERIFY CHECKSUM` is used for the in-memory data page checksum activity. Those pages that have been written to stable media with a CHECKSUM status are eligible for in-memory checksumming if the dynamic trace flag `-T831` is enabled. The data page must have received the initial checksum value during a write to stable media to participate in the in-memory checksum operations.

To reduce the performance affect, the in-memory checksum is only audited during certain page state transitions. The key page states that trigger in-memory checksumming are outlined after the table. The following table describes the states that a page can be in.

Page State	State Description
Dirty	The data page is considered to be dirty when the page has been modified and has not been written to stable media. As soon as a dirty page is saved (written) to stable media, it is considered to be clean.

Clean	The data page is considered to be clean or a constant page when it is the same image as that stored on stable media.

In-memory checksum auditing occurs on a data page when the following conditions are true:

The page was written to disk when `PAGE_VERIFY CHECKSUM` was enabled.

The `PAGE_VERIFY CHECKSUM` option is enabled for the database.

Trace flag `-T831` is enabled.

The `PAGE_VERIFY` actions continue to occur during the read and write of data pages. The in-memory checksumming occurs when `-T831` is enabled. The following table outlines the checksum actions that occur when the database is correctly enabled for CHECKSUM protection, the page contains a valid checksum, and trace flag `-T831` is enabled.

Action	Description
Page Read	**Page State** **State Description** **Physical Read** Checksum is validated as soon as the read finishes; the checksum is retained for in-memory validation. **Logical Read** No in-memory checksum auditing occurs.

Page Modification Request	Page State State Description
	Dirty As soon as a page has been dirtied, the checksum is no longer maintained. The checksum will be recalculated when the page is written to stable media. No in-memory checksum auditing occurs. **Clean** The transition from clean to dirty triggers in-memory checksum validation. A failure during this transition indicates that the page was damaged during a period in which it was considered to be read-only.

Discard	A page is termed ‘discarded’ when it is returned to the free list. **Page State** **State Description** **Dirty** A dirty page cannot be discarded. The page must first be written to stable media and returned to a clean state before it can be discarded. **Clean** The act of discarding a clean page triggers in-memory checksum validation. A failure during this transition indicates that the page was damaged during a period in which it was considered to be read-only. **Note:** Pages that are never modified (never dirtied) remain in the clean state until they are discarded at which time the checksum is validated for the constant page.

For added, always on, protection the lazy writer performs clean (constant) buffer checksum validations. This is always on and does not require that `-T831` be enabled. Every second the lazy writer updates the buffer pool performance counters and performs various housekeeping activities. During this housekeeping, the lazy writer sweeps over 16 buffers. When the lazy writer finds a clean buffer with a valid checksum, it validates the checksum. If a failure is detected, an 832 error message is logged. This is used as a low affect, background, in-memory checksum audit activity. Pages that remain in a clean (constant) state for lengthy periods enable the lazy writer audit to catch unexpected damage before the page is discarded.

If the audit check fails, SQL Server error message `832` is reported to indicate that the error condition was detected. If you are encountering in-memory checksum failures, perform the following diagnostics.

Test backups to make sure that the restore strategy remains correctly intact.

Perform full hardware testing, focusing specifically on memory components.

Review any third-party products installed on the system or that are used in the SQL Server process space. Third-party components could scribble and cause problems. Such components could be COM objects, extended stored procedures, Linked Servers, or other entities.

Make sure that all operating system fixes are applied to the server.

Make sure that any virus protection is up to date and the system is free of viruses.

Review the location of the page file for SQL Server I/O compliance requirements.

Enable latch enforcement as described later in this document to help isolate the source of the damage.

Try to use the same input buffers or replay a SQL Server Profiler trace to reproduce the problem. If a reproduction is obtained, can it be reproduced on another computer? If it can be reproduced, contact Microsoft SQL Server Support for additional assistance.

Latch enforcement

SQL Server 2000 and SQL Server 2005 can perform latch enforcement for database pages located on the buffer pool cache. Latch enforcement changes the virtual memory protection (`VirtualProtect`) as the database pages are transitioned between the clean and dirty states. The following table outlines the virtual protection states.

Page State	Virtual Protection State
Dirty	Read Write during the modification.
Clean	Read Only; any attempt to modify the page when this protection is set (termed a scribbler) causes a handled exception, generating a mini-dump for additional investigation.

The database page remains in the virtual protection state of Read Only until the modification latch is acquired. When the modification latch is acquired. the page protection is changed to Read Write. As soon as the modification latch is released, the page protection is returned to Read Only.

> **Note**: The default latch enforcement protection setting is disabled. Latch enforcement may be enabled with trace flag –T815. SQL Server 2000 SP4 and 2005 allow for the trace flag to be enabled and disabled without a restart of the SQL Server process by using the `DBCC traceon(815, -1)` and `DBCC traceoff(815,-1)` commands. Earlier versions of SQL Server require the trace flag as a startup parameter.
>
> **Note**: The trace flag should only be used under the direction of Microsoft SQL Server Support as it can have significant performance ramifications and virtual protection changes may not be supported on certain operating system versions when you are using PAE/AWE.
>
> **Note:** Windows extended support for `VirtualProtect` in Windows Server™ 2003 SP1 and Windows XP SP2 to allow virtual protection of AWE allocated memory. This is a very powerful change but could affect the performance of SQL Server if it is configured to use AWE or locked pages memory due to the extended protection capabilities.

Latch enforcement applies only to database pages. Other memory regions remain unchanged and are not protected by latch enforcement actions. For example, a TDS output buffer, a query plan, and any other memory structures remain unprotected by latch enforcement.

To perform a modification, SQL Server must update the page

protection of a database page to Read Write. The latch is used to maintain physical stability of the database page so the modification latch is only held for long enough to make the physical change on the page. If the page is damaged during this window (scribbled on), latch enforcement will not trigger an exception.

In versions earlier than SQL Server 2004 SP4, SQL Server latch enforcement protection involved more protection transitions. The following table outlines the protection transactions performed by SQL Server earlier than SQL Server 2000 SP4.

Page State	Virtual Protection State
Dirty	Read Write during the modification.
Clean No References	No Access; any attempt to read or write from the page causes an exception.
Clean With References	Read Only; any attempt to modify the page when this protection is set (termed a 'scribbler') causes a handled exception, generating a mini-dump for additional investigation.

Because virtual protection transitions are expensive, SQL Server 2000 SP4 and SQL Server 2005 no longer transition the page protection to No Access, thereby reducing the number of transitions significantly. The older implementation could raise an exception for an invalid read try where the newer implementations cannot. The overhead of No Access protection transitions frequently made latch enforcement too heavy for use in a production environment. Leaving the page with Read Only access reduces the number of protection changes significantly and still helps in the identification of a scribbler.

SQL Server does not return all data pages to Read Write protection as soon as the trace flag is disabled. The pages are returned to Read Write protection as they are modified so that it may take some time to return to a fully non-latch enforced buffer pool.

Checksum on backup and restore

SQL Server 2005 `BACKUP` and `RESTORE` statements provide the `CHECKSUM` option to include checksum protection on the backup stream and trigger the matching validation operations during restore. To achieve a checksum-enabled backup, the `BACKUP` command must include the `CHECKSUM` option.

The backup and restore processes try to work with large blocks of data whenever possible. For example, the backup operation examines the allocation bitmaps in the database to determine what data pages to stream to the backup media. As soon as a block of data is identified, the backup operation issues a large 64 KB to 1 MB read from the data file and a matching write operation to the backup stream. The backup

operation avoids touching individual bytes of the data pages or log blocks to maximize its throughput as a high speed copy implementation.

Backup and restore operations that use checksum capabilities increase data integrity protection and also increase CPU usage requirements. A backup or restore with the checksum option requires that each byte be interrogated as it is streamed, thereby increasing CPU usage. The checksum that is used for backup and restore uses the same algorithm to calculate the checksum value for the backup media as is used for data pages and log blocks.

The following rules apply to the BACKUP and RESTORE command CHECKSUM operations.

By default, SQL Server 2005 BACKUP and RESTORE operations maintain backward compatibility (NO_CHECKSUM is the default).

The database's PAGE_VERIFY setting has no affect on backup and restore operations; only the CHECKSUM setting on the backup or restore command is relevant.

The backup and restore checksum is a single value representing the checksum of the complete stream; it does not represent individual pages or log blocks located in the backup stream. The value is calculated during the backup and stored with the backup. The value is recalculated during the restore and checked against the stored value.

Backup with the CHECKSUM option will not change the pages as it saves them to the backup media; a page's protection state (NONE, CHECKSUM, or TORN) is maintained as read from the database file. If a checksum was already stored on the data page, it is verified before the page is written to the backup stream.

Restore and Verify commands can be used to validate the CHECKSUM if the backup was created by using the CHECKSUM option. Trying to restore with the CHECKSUM option on a backup without a checksum returns an error.

For more information on backup and restore, see SQL Server 2005 Books Online.

Page-level restore

SQL Server 2005 Enterprise Edition introduces page-level restore to repair damaged pages. The database can restore a single page from backup instead of requiring a full database, file group, or file restore. For complete details, see SQL Server 2005 Books Online.

Database available during Undo phase

SQL Server 2005 Enterprise Edition enables access to the database as soon as the Redo phase of recovery is finished. Locking mechanisms are used to protect the rollback operations during the Undo phase. To reduce downtime, page-level restore can be combined with the crash recovery capability of enabling access to the database during the Undo phase of recovery.

Torn page protection

Torn page protection has not significantly changed from SQL Server 7.0 and SQL Server 2000. This section provides details on torn page protection and how it works to help you compare TORN protection and CHECKSUM protection. A torn page commonly indicates

that one or more sectors have been damaged.

Common reasons

Following are some common problems found by Microsoft SQL Server Support that cause TORN page error conditions.

The subsystem or hardware does not handle the data correctly and returns a mix of sector versions. This has been reported on various controllers and firmware because of hardware read-ahead cache issues.

Power outages occur.

Bit flips or other damage occurs on the page header. This indicates that a page status of TORN detection was enabled when really was not.

Implementation

Torn page protection toggles a two-bit pattern between 01 and 10 every time the page is written to disk. Write A obtains bit protection of 01 and write B obtains bit protection of 10. Then write C obtains 01 and so on. The low order (last) two bits of each 512-byte sector are stored in the page header and replaced with the torn bit pattern of 01 or 10.
The relevant members of the SQL Server data page header are shown in the following list together with a TORN bit layout diagram.

Member	Description
m_flagBits	Bit field where TORN, CHECKSUM or NONE is indicated.
m_tornBits	Contains the TORN or CHECKSUM validation value(s).

Figure 1

The torn page toggle bits are established as 01 or 10 and positioned in the low order 2 bits of the `m_tornBits` value. For the remaining 15, 512-byte sectors, the low order 2 bits of each sector are positioned in incrementing bit positions of the m_tornBits and the established bit pattern is stored in their location.
Following are the steps shown in the previous diagram.

Step #1: **The original sector bit values are stored in the m_tornBits from low order to high order (like a bit array), incrementing the bit storage positions as sector values are stored.**

Step #2: **The established torn bit pattern is stored in the low order two bits of each sector, replacing the original values.**

When the page is read from disk and `PAGE_VERIFY` protection is enabled for the database, the torn bits are audited.

Step #1: **The low order bits of the `m_tornBits` are checked for the**

pattern of either 10 or 01 to make sure that the header is not damaged.

Step #2: **The low order two bits in each sector are checked for the matching torn bit pattern as stored in the low order two bits of `m_tornBits`.**

If either of these checks fail, the page is considered TORN. In SQL Server 2000 this returns an 823 error and in SQL Server 2005 it gives you an 824 error.

Step #3:

SQL Server 2000: **Replaces the original values as each sector is checked, even if an error was detected. This makes it difficult to investigate which sector was torn.**

SQL Server 2005: **Enhances troubleshooting by leaving the bits unchanged when an error is detected. Investigate the page data to better determine the torn footprint condition.**

Stale read protection

Stale reads have become a problem that is frequently reported to Microsoft SQL Server Support. A stale read occurs when a physical read returns an old, fully point-in-time consistent page so that it does not trigger TORN or CHECKSUM audit failures; instead, the read operation returns a previous data image of the page. This is also called a *lost write* because the most recent data written to stable media is not presented to the read operation.

A common cause of stale reads and lost writes is a component such as a hardware read-ahead cache that is incorrectly returning older cached data instead of the last write information.

This condition indicates serious I/O subsystem problems leading to page linkage corruption, page allocation corruption, logical or physical data loss, crash recovery failures, log restore failures, and a variety of other data integrity and stability issues.

In a SQL Server 2000 SP3-based hot fix (build 8.00.0847), stale read detection was added. This addition is outlined in the Microsoft Knowledge Base article, HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;826433" PRB: Additional SQL Server Diagnostics Added to Detect Unreported I/O Problems (HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;826433" http://support.microsoft.com/default.aspx?scid=kb;en-us;826433).

Enhancements

By changing from a ring buffer to a hash table design, SQL Server 2000 SP4 and SQL Server 2005 provide enhanced, low-overhead stale read checking. The original SQL Server 2000 SP3 implementation only checks for a stale condition if another error was found first (605, 823, and so forth). The hash table design, used in newer builds, allows for the page read sanity checking to include a stale read check when trace flag `-T818` is enabled for any page that is read without a noticeable performance affect.

For SQL Server 2005, every time a page is written to disk, a hash table entry is inserted or updated with the DBID, PAGEID, RECOVERY UNIT, and LSN that is being flushed to stable media. When a read is complete, the hash table is searched for a matching entry. The matching DBID and PAGEID entry is located. The hash table LSN value is compared to the LSN value that is stored in the page header. The LSN values must match or the page is considered damaged. Thus, if the most recent LSN that was written was not returned during the subsequent read operation, the page is considered damaged.

To maintain a high level of I/O performance and limit the memory footprint, the hash table size is bounded. It tracks only the recent window of data page writes. The number of I/Os tracked varies between the 32- and 64-bit versions of SQL Server 2000 SP4 and SQL Server 2005. To optimize speed, each hash bucket and its associated entries are designed to fit in a single, CPU cache line, thereby limiting the hash chain length to five entries for each bucket. In 32-bit installations, the total size of the hash table is limited to 64 KB (equating to 2,560 total entries = 20-MB window of data) and on 64-bit installations to 1 MB (equating to 40,960 total entries = 320-MB window of data).

The size restriction is based on the testing of known bugs that caused stale reads or lost writes. The bug characteristics typically involve a hardware memory cache that held the older page data and a read operation that immediately followed or overlapped the write operation.

Stalled I/O detection

Database engine performance can be highly affected by the underlying I/O subsystem performance. Stalls or delays in the I/O subsystem can cause reduced concurrency of your SQL Server applications. Microsoft SQL Server Support has experienced an increase in I/O subsystem

delays and stall conditions resulting in decreased SQL Server performance capabilities.

For a SQL Server 2000 installation, an I/O stall or delay is frequently detected by watching `sysprocesses` for I/O-based log and/or buffer (data page) wait conditions. Whereas small waits might be expected, some filter drivers or hardware issues have caused 30+ second waits to occur, causing severe performance problems for SQL Server-based applications.

Starting with SQL Server 2000 SP4 and SQL Server 2005, SQL Server monitors and detects stalled I/O conditions that exceed 15 seconds in duration for data page and log operations. The following Microsoft Knowledge Base article describes the SQL Server 2000 SP4 implementation: HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;897284" SQL Server 2000 SP4 diagnostics help detect stalled and stuck I/O operations (HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;897284" http://support.microsoft.com/default.aspx?scid=kb;en-us;897284).

SQL Server 2000 SP4 and SQL Server 2005 also increase the visibility of latch operations. A latch is used to guarantee the physical stability of the data page when a read from or a write to stable media is in progress. With the increased latch visibility change, customers are frequently surprised after they apply SQL Server 2000 SP4 when a SPID appears to block itself. The following article describes how the latch information displayed in sysprocesses can be used to determine I/O stall conditions as well as how a SPID can appear to block itself: HYPERLINK "http://support.microsoft.com/kb/906344/en-us" The blocked column in the sysprocesses table is populated for latch waits after you install SQL Server 2000 SP4 (**HYPERLINK "http://support.microsoft.com/kb/906344/en-us"** http://support.microsoft.com/kb/906344/en-us).

SQL Server 2005 contains the stalled I/O monitoring and detection. The stalled I/O warning activity is logged when a stall of 15 seconds or longer is detected. Additionally, latch time-out error messages have been extended to clearly indicate that the buffer is in I/O. This indicates that the I/O has been stalled for 300 seconds (five minutes) or more.

There is a clear difference between *reporting* and *recording*. Reporting only occurs in intervals of five minutes or longer when a new I/O action occurs on the file. Any worker posting an I/O examines the specific file for reporting needs. If I/O has been recorded as stalled and five minutes has elapsed from the last report, a new report is logged to the SQL Server error log.

Recording occurs in the I/O completion routines, and the lazy writer checks all pending I/Os for stall conditions. Recording occurs when an I/O request is pending (FALSE == HasOverlappedIoCompleted) and 15 seconds or longer has elapsed.

> **Note:** The `FALSE` return value from a `HasOverlappedIoCompleted` call indicates that the operating system or I/O subsystem has not completed the I/O request.

sys.dm_io_pending_io_requests (DMV)

SQL Server 2005 provides dynamic access to pending I/O information so that a database administrator can determine the specific database, file, and offset leading to a stall. The dynamic management view (DMV) **sys.dm_io_pending_io_requests** contains details about the

offset and status of each outstanding I/O request. This information can be used by Microsoft Platforms Support and various utilities to track down the root cause. For more information, go to HYPERLINK "http://support.microsoft.com" http://support.microsoft.com and search for information related to IRP and ETW event tracing.

The `io_pending` column is a specific key for evaluating the result set. The `io_pending` column indicates whether the I/O request is still pending or if the operating and I/O subsystem have completed it. The value is determined by using a call to `HasOverlappedIoCompleted` to determine the status of the I/O request. The following table outlines the returned value possibilities for `io_pending`.

io_pendingValue	Description
TRUE	Indicates the asynchronous I/O request is not finished. SQL Server is unable to perform additional actions against the data range until the operating system and I/O subsystem complete the I/O request. To learn more about pending I/O conditions, see `HasOverlappedIoCompleted` in the SDK documentation. Lengthy asynchronous I/O conditions typically indicate a core I/O subsystem problem that should be addressed to return SQL Server to ordinary operating conditions.

FALSEIndicates the I/O request is ready for additional processing actions by SQL Server.

If the pending time of the I/O continues to climb, the issue may be a SQL Server scheduling problem. For a discussion of SQL Server scheduler health and associated troubleshooting, see the following white paper.

HYPERLINK "http://download.microsoft.com/download/4/f/8/4f8f2dc9-a9a7-4b68-98cb-163482c95e0b/DiagandCorrectErrs.doc" How to Diagnosis and Correct Errors 17883, 17884, 17887, and 17888 (HYPERLINK "http://download.microsoft.com/download/4/f/8/4f8f2dc9-a9a7-4b68-98cb-163482c95e0b/DiagandCorrectErrs.doc" \t "_blank" http://download.microsoft.com/download/4/f/8/4f8f2dc9-a9a7-4b68-98cb-163482c95e0b/DiagandCorrectErrs.doc)

The `io_pending_ms_ticks` column is the elapsed milliseconds (ms) of the I/O request that was posted to the operating system.

The `io_handle` is the file HANDLE that the I/O request is associated with. This column can be joined to the dynamic management function (DMF) **sys.dm_io_virtual_file_stats** column `file_handle` to obtain specific file and database association from the I/O. The following is an example of a query to obtain this information.

```
SELECT fileInfo.*, pending.*
    FROM sys.dm_io_pending_io_requests AS pending
INNER JOIN (SELECT * FROM sys.dm_io_virtual_file_stats(-1,
-1))
```

```
    AS fileInfo ON fileInfo.file_handle = pending.io_handle
```

This join can be additionally enhanced by adding information such as the database name or by using the offset to calculate the actual PAGEID; (`Offset/8192 = PAGEID`).

> **WARNING:** DMVs and DMFs access core system structures to produce the result set. Internal structures must be accessed with thread safety, which may have performance ramifications. The use of DMVs that access core SQL Server components should be limited to avoid possible performance affects.

Read retry

SQL Server 2005 extends the use of read retry logic for data pages to increase read consistency possibilities. A read retry involves performing exactly the same read operation immediately following a read failure in an attempt to successfully complete the read.

Microsoft has successfully used read retries to compensate for intermittent failures of an I/O subsystem. Read retries can mask data corruption issues in the I/O subsystem and should be investigated carefully to determine their root cause. For example, a disk drive that is going bad may intermittently return invalid data. Re-reading the same data may succeed and the read retry has provided runtime consistency. However, this is a clear indicator that the drive is under duress and should be examined carefully to avoid a critical data loss condition.

The Microsoft Exchange Server product added read retry logic and has experienced improved read consistency. This section outlines how and when SQL Server 2000 and SQL Server 2005 perform read retry operations.

Resource-based retries

SQL Server 2000 performs read retries only when beginning the read operations fails and returns an operating system error of ERROR_WORKING_SET_QUOTA (1453) or ERROR_NO_SYSTEM_RESOURCES (1450). Unlike the SQL Server 2005 enhancements, SQL Server 2000 does not try any other form of read retry other than sort failures.

When the error occurs, the SQL Server worker yields for 100ms and tries the read operation again. This loop continues until the I/O is successfully issued. A simplified example demonstrates this behavior.

```
WHILE(    FALSE == ReadFile()
    && (1450 == GetLastError() ||  1453 == GetLastError())
)
```

```
{
    Yield(100);
}
```

SQL Server 2005 maintains the same logic when it is trying to start a read operation.

Sort retries

SQL Server 7.0, 2000, and 2005 have sort-based retry logic. These frequently appear as 'BobMgr' entries in the SQL Server error log. When a read of a spooled sort buffer from **tempdb** fails, SQL Server tries the read again. The retries are only attempted several times before they are considered to be fatal to the sort. Sort retries should be considered a serious I/O stability problem. To correct the problem, try moving **tempdb** to different location.

Other read failure retries

SQL Server 2005 has extended the read retry logic to read failures that occur after the read was successfully started. When `ReadFile` returns `TRUE`, this indicates that the operating system accepted the application's request to read from the file. If a subsequent failure is experienced, the result is a SQL Server error such as an 823 or 824.

SQL Server 2005 allows for read retry operations to continuously occur when the read finishes with an error caused by a resource shortage. For all non-resource shortage conditions, four (4) more retries may be tried.

Each successive retry yields before it tries the read operation again. The yield is based on the following formula: (`yield time = retry attempt * 250ms`). If the error condition cannot be resolved within four retries (five total times: one initial read and four retries), an 823 or 824 error is reported. SQL Server 2005 saves the original error condition details, such as a checksum failure. It also includes the original details with the error report in the SQL Server error log.

If the retry succeeds, an informational message is added to the SQL Server error log. This indicates that a retry occurred. The following is an example of the message.

> "A read of the file `<<FILE NAME>>` at offset `<<PHYSICAL OFFSET>>` succeeded after failing `<<RETRY COUNT>>` time(s) with error: `<<DETAILED ERROR INFORMATION>>`. Additional messages in the SQL Server error log and system event log may provide more detail. This error condition threatens database integrity and must be corrected. Complete a full database consistency check (DBCC CHECKDB). This error can be caused by many factors; for more information, see SQL Server Books Online."

Read retry problems are a serious problem with data stability as the I/O subsystem is returning incorrect data to SQL Server. This condition is likely to cause a fatal SQL Server error or even a system-wide failure. Additionally, the retry activity has performance affect on SQL Server operations. This is because as soon as a read error is detected, the worker performs the reties until it succeeds or the retry limit is exhausted.

Page audit

SQL Server 2000 SP4 and SQL Server 2005 include additional page audit capabilities. Enabling the dynamic trace flag `-T806` causes all physical page reads to run the fundamental DBCC page audit against the page as soon as the read is complete. This check is performed at the same point as the `PAGE_AUDIT` and other logical page checks are performed.

This is another way to locate data page corruption in areas of the page other than the basic page header fields, which are always checked during the physical read. For example, when trace flag `-T806` is enabled, the row layout is audited for appropriate consistency.

Page audit was first added in SQL Server 2000 SP3, hot fix build 8.00.0937. For more information, see the following Microsoft Knowledge Base article: HYPERLINK "http://support.microsoft.com/kb/841776/en-us" FIX: Additional diagnostics have been added to SQL Server 2000 to detect unreported read operation failures (HYPERLINK "http://support.microsoft.com/kb/841776/en-us" http://support.microsoft.com/kb/841776/en-us).

> **Note:** Enabling page audit capabilities can increase the CPU load on the server and decrease overall SQL Server performance.

SQL Server 2005 introduces checksum protection, which generally supersedes page audit capabilities. Checksum protection ensures that every bit on the page is the same as that written to stable media.

For SQL Server 2005 installations, checksum is often a better solution than constant data integrity auditing. However, page audit can help catch corruption which was stored to stable media even though physical page consistency was not compromised. Microsoft SQL Server Support has encountered an example of this. In that instance, a third-party extended stored procedure scribbled on a data page that was already marked dirty. The checksum was calculated on a damaged page and the page was written. The reading in of this page, with page audit enabled, could indicate the error condition when checksum would not detect the failure. If you believe the server may be experiencing a problem that checksum cannot detect but page audit is detecting, consider in-memory checksumming and latch enforcement to help locate the scribbler.

Log audit

SQL Server 2000 and 2005 include trace flag –T3422 which enables log record auditing. Troubleshooting a system that is experiencing problems with log file corruption may be easier using the additional log record audits this trace flag provides. Use this trace flag with caution as it introduces overhead to each transaction log record.

Checkpoint

SQL Server 2005 implemented user-controlled I/O target behavior for the manual `CHECKPOINT` command and improved the I/O load levels during automatic checkpointing. For more information on how to issue a manual `CHECKPOINT` command and specify a target value (in seconds), see SQL Server 2005 Books Online.

Microsoft has received requests to implement a more dynamic checkpoint algorithm. For example, this would be useful during a SQL Server shutdown. Especially for highly available databases, a more aggressive checkpoint can reduce the amount of work and time that crash recovery needs during a restart.

The amount of transaction log activity determines when to trigger

checkpoint of a database. Transaction log records have a recovery cost value calculated in milliseconds. Each time a new transaction log record is produced, the accumulated cost is used to determine the estimated recovery time required since the last checkpoint. When the recovery time goal is exceeded, a checkpoint is triggered. This keeps the crash recovery runtime within the specified `recovery interval` goal.

The following base rules apply to checkpoint. The term *latency* as it is used here indicates the elapsed time from when the write was issued until the write is considered complete by the checkpoint process.

Action	Description
Manual Checkpoint – Target Specified	I/O latency target set to the default of 20ms. The target is set to 100ms if shutdown is in progress. Maximum number of standing I/Os is capped at the larger of the following calculations: Committed Buffer Count / 3750 80 * Number of Schedulers Number of outstanding I/Os is constantly adjusted so that progress through the buffer pool is commensurate with elapsed time and target time.
Manual Checkpoint – No target specified **- or -** **Automatic Checkpoint in response to database activity**	I/O latency target set to the default of 20ms. The target is set to 100ms if shutdown is in progress. Maximum number of standing I/Os is capped at the larger of the following calculations: Committed Buffer Count / 3750 80 * Number of Schedulers Minimum number of outstanding I/Os required is 2. Number of outstanding I/Os is adjusted to keep write response time near latency target.

For any checkpoint invocation

When checkpoint reaches its outstanding I/O target, it yields until one of the outstanding I/Os is finished.

For no target specified or automatic checkpointing

The checkpoint process tracks checkpoint-specific I/O response times. It can adjust the number of outstanding I/O requests if the I/O latency of checkpoint writes exceed the latency target. As checkpoint processing continues, the goal for the outstanding number of I/Os is adjusted in order to maintain response times that do not exceed the established latency goal. If the outstanding I/O levels begin to exceed the tolerable latency goals, checkpoint adjusts its activity to avoid possible affects on the overall system.

For a manual checkpoint, target specified

When checkpoint processing detects that it is ahead of the specified target, it yields until activities fall within goal, as outlined in the previous table, or until all the outstanding checkpoint I/Os finish. If all outstanding I/Os are complete, checkpoint issues another I/O and again tries to use the target goal.

SQL Server 2005 SP1 allows for continuous check pointing

SQL Server 2005 SP1 alters the checkpoint algorithm slightly. SQL Server 2005 does not add time between I/Os. Therefore, the checkpoint process may finish ahead of the target schedule. Service Pack 1 introduces the appropriate delays to honor the target as closely as possible. We do not recommend this, but administrators can disable automatic checkpointing and use manual checkpointing with a specified target. Putting the manual, targeted checkpoint in a continuous loop provides a continuous checkpoint operation. Do this with extreme caution because checkpoints are serialized and this could affect other databases and backup operations. It also requires that a checkpoint process be established for all databases.

Notice that SQL Server 2005 Service Pack 1 also contains a fix for a very rare checkpoint bug. The fix is for a very small window where checkpoint could miss flushing a buffer. This could lead to unexpected data damage. To avoid this problem, apply SQL Server 2005 SP1.

WriteMultiple extended

SQL Server 7.0 introduced an internal routine named `WriteMultiple`. The `WriteMultiple` routine writes data pages to stable media. For more information, see"Flushing a Data Page To Disk" in HYPERLINK "http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlIObasics.mspx" SQL Server I/O Basics on MSDN.

SQL Server 7.0 and 2000 could issue a WriteMultiple operation for up to 16 pages (128 KB). SQL Server 2005 extends the capability of WriteMultiple up to 32 pages (256 KB). This may change the block size configurations for your performance goals. For more information about physical database layout, see the article " HYPERLINK "http://download.microsoft.com/download/4/f/8/4f8f2dc9-a9a7-4b68-98cb-163482c95e0b/PhysDBStor.doc" Physical Database Storage Design" (HYPERLINK "http://download.microsoft.com/download/4/f/8/4f8f2dc9-a9a7-4b68-98cb-163482c95e0b/PhysDBStor.doc" http://download.microsoft.com/download/4/f/8/4f8f2dc9-a9a7-4b68-98cb-163482c95e0b/PhysDBStor.doc).

SQL Server 2005 varies the `WriteMultiple` logic. In SQL Server 7.0 and 2000, the function accepts a starting page ID. The starting page and up to 16 subsequent, contiguous dirty pages for the same database are bundled in a single write request. SQL Server does this by using hash table lookups for subsequent contiguous pages. When a page is not found or a page is found that is clean, the I/O request is considered finished.

SQL Server 2005 adds additional lookup and safety steps to `WriteMultiple`. SQL Server 2005 does the forward page search in the same way as SQL Server 7.0 and 2000. When the forward search is finished, SQL Server 2005 can do a backward search if all 32 pages of the I/O request are not yet filled. The same hash table lookup activity

occurs when SQL Server searches for more pages. For example if WriteMultiple was passed page 1:20, the search would examine 1:19, 1:18, and so on. The search continues until:

A page is not found.

A page is found to be clean.

All 32 pages for the I/O have been identified.

SQL Server 2005 adds additional page header checks. One such additional check is the actual page ID check. The expected page ID is compared to that of the actual page header. The prevents writing a scribbled or incorrect page to disk and causing permanent database damage.

Read-ahead enhanced

In SQL Server 2005, the read-ahead design is enhanced so that it reduces physical data transfer requirements by trimming the leading and trailing pages from the request if the data page(s) are already in the buffer pool.

For more information on SQL Server read-ahead logic, see HYPERLINK "http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlIObasics.mspx" SQL Server I/O Basics (HYPERLINK "http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlIObasics.mspx" http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlIObasics.mspx).

For example, a read-ahead request is to be issued for pages 1 through 128 but pages 1 and 128 are already located in the SQL Server buffer pool. The read-ahead request would be for pages 2 through 127 in SQL Server 2005. In comparison, SQL Server 2000 requests pages 1 through 128 and ignores the data that is returned for pages 1 and 128.

Sparse files / Copy on write / Streams

NTFS sparse file technology is used for database snapshots and online `DBCC CHECK*` operations. This section provides more detailed information about this technology in SQL Server.

Note: At the time of publication, manufacture-specific "thin provisioning" implementations have not yet been tested. See the SQL Server Always On Storage Solution Review program (HYPERLINK "http://www.microsoft.com/sql/AlwaysOn" http://www.microsoft.com/sql/AlwaysOn) for newer information about this topic.

Streams

Online `DBCC CHECK*` uses a transient, sparse file stream for each data file that is checked. The streams are named using the following template: "`<<ORIGINAL FILE>>:MSSQL_DBCC<<DBID>>`". The stream is a secondary data area associated with the original file provided by the file system. Online DBCC uses the stream to create a transient snapshot of the database as long as it performs checks. This snapshot is unavailable to database users. The snapshot stream enables online DBCC to create and test all facts against an exact point-in-time replica of the database. It requires only limited physical storage to do this.

During online DBCC, only those pages that are modified after DBCC started are stored in the stream. When online DBCC is finished, the stream is deleted.

It is worthy to notice that the stream enables DBCC to reduce its locking granularity. If the stream cannot be created or runs out of space, DBCC reverts to the older, TABLE LOCK behavior. Administrators should review the free space available on each volume to make sure that high database concurrency is maintained.

For more information about online DBCC, see DBCC Internal Database Snapshot Usage in SQL Server 2005 Books Online.

Copy-on-write and sparse files

Snapshot databases contain images of data pages that have been modified after they were created. Establishing a database snapshot includes performing an immediate, secondary rollback of active transactions at the time of creation. Active transactions in the primary database must be rolled back in the new snapshot to obtain the correct point in time. Transaction states in the primary database remain unaffected.

To conserve physical disk space, snapshot databases are stored on sparse files. This limits the physical disk space requirement of the snapshot database to that of the modified images. As more data pages are modified in the parent database, the physical storage requirement of the snapshot database increases.

Snapshot database files are named as specified by the CREATE DATABASE command. For example, the snapshot parent database can contain the main.mdf file and the snapshot may use snapshot_main.mdf.

SQL Server 2005 implements copy-on-write for the data pages of a snapshot database. Before the primary database page can be modified, the original data page is written to any associated database snapshot. It is important for administrators to monitor the physical size of the snapshot databases to determine and predict the physical storage requirements. Notice that the smallest allocation unit in a sparse file is 64 KB. Therefore, the space requirement may grow faster than you expect.

For more information about snapshot databases, see How Database Snapshots Work in SQL Server 2005 Books Online.

Recovery redo and undo operations use the transaction log to return a database to a consistent transactional state. However, this logic does not apply to a snapshot database. Notice that snapshot databases do not have transaction logs. Therefore, the copy-on-write activity must be complete before the primary database transaction can continue.

The snapshot uses a combination of API calls to determine the 64-KB allocated regions in the sparse file. It must also implement an in-

memory allocation bitmap to track the individual pages that have been stored in the snapshot. During the creation or expansion of the snapshot file, SQL Server sets the appropriate file system attributes so that all unwritten regions return complete zero images for any read request.

When a modification request is started in the primary database, the data page may be written to any snapshot database and the in-memory allocation bitmap is appropriately maintained. Because it can introduce I/O on the snapshot database when the copy-on-write decision is being made, it is important to consider the additional overhead. Determining when a copy of the database page is necessary involves the combination of various API calls and possible read requests. Therefore, read requests may be generated to the snapshot database in order to determine whether the data page has already been copied to the snapshot database.

When SQL Server performs the copy-on-write operation into a sparse file, the operating system may perform the write in a synchronous manner when it acquires new physical storage space. To prevent the synchronous nature of these writes from affecting the SQLOS scheduler, the copy-on-write writes may be performed by using secondary workers from the worker pool. In fact, if multiple snapshots exist for the same primary database, the writes can be performed by using multiple workers in a parallel manner. The initiating worker waits for the secondary workers and any writes to complete before continuing with the modification on the original data page. When SQL Server waits for the writes to complete, the originating worker may show a wait status such as replica write or a latch wait for the replica data page which is in I/O.

Even if the transaction is rolled back, the data page has been written to the snapshot. As soon as a write occurs, the sparse file obtains the physical storage. It is no longer possible to fully roll back this operation and reclaim the physical disk space.

> **Note**: If a copy-on-write operation fails, the snapshot database is marked as suspect. SQL Server generates the appropriate error.

Realize that the increased copy-on-write operations (actual writes or reads to determine whether a copy-on-write is necessary) can change the performance dynamics of some queries. For example, the I/O speed of a snapshot database could limit certain query scalabilities. Therefore, you should locate the snapshot database on a high speed I/O subsystem.

> **Note:** Utilities such as WinZip, WinRAR, copy and others do not maintain the actual integrity of a sparse file. When a file is copied by using these utilities, all unallocated bytes are read and restored as zeros. This requires actual allocations and the restored file looses the sparse file attributes. To copy a sparse file, you must use a utility that understands how to capture and restore metadata

structure of the file.

Stream and sparse file visibility

Stream and sparse file sizes are easier to monitor if you are aware of some key visibility limitations.

Secondary file streams are frequently invisible to commands such as '`dir`'. This can make it difficult to determine how much copy-on-write activity has occurred during an online DBCC CHECK*. However, various third-party utilities are available which can be used to view size information about file streams.

Similarly, sparse file sizes that are reported to commands such as '`dir`' indicate the complete file size as established by the End Of File (EOF) setting and not the actual physical storage on disk. Windows Explorer shows the logical as 'Size' and the physical as 'Size on Disk.'

When SQL Server writes to a sparse database file, it can acquire space in database extent sizes of 64 KB (8 pages * 8KB each = 64KB). SQL Server 2005 detects when a segment of the file has not been allocated in the sparse file. It not only copies the page during the copy-on-write operation but establishes the next seven pages (one extent) with all zero images. This reduces the operating system-level fragmentation by working on the common operating system file system cluster boundaries. It also enables future copy-on-write requests for any one of the other seven pages to finish quickly because the physical space and file system tracking has already been established. It also enables SQL Server to use the file level allocation information to determine what extents are physically allocated in the sparse file. The zeroed page images can be used to determine which of the pages that are enclosed in allocation region have been copied from the primary database.

Snapshot reads

SQL Server 2005 is designed to read data from both the snapshot file and the matching primary database file when data is queried in the snapshot database. By using the in-memory sparse database allocation bitmaps and the zero page images, SQL Server can quickly determine the actual location of data pages that are required by a query.

For example, during query execution SQL Server can elect to perform a large read, reading in eight or more data pages with a single read request such as a read-ahead. Look at a specific example.

A read-ahead operation in the primary database, for the same region, creates a single read of 64 KB to retrieve eight contiguous pages. However, if the third and sixth pages have been modified and copied (with copy-on-write) to the snapshot database, this causes a split set of reads. This example requires five separate read requests.

Pages 1 and 2 can be read from the primary

Page 3 from the snapshot

Pages 4 and 5 from the primary

Page 6 from the snapshot

Pages 7 and 8 from the primary

SQL Server always tries to optimize physical data access requests but a query against the snapshot database may have to perform more I/Os than the identical query executed in the primary database.

WARNING: If you use SQL Server database snapshots or online DBCC CHECK* operations, apply the operating system for system bug 1320579 fix to avoid corruption of the snapshot. This is covered in the following article: Error message when you run the DBCC check command in SQL Server 2005: "8909 16 1 Table error: Object ID 0, index ID -1, partition ID 0, alloc unit ID 0 (type unknown)" (909003)

Instant file initialization

Newer operating system versions, including Windows XP and Windows Server 2003, implement instant file initialization capabilities by supplying the API `SetFileValidData`. This API enables SQL Server to acquire physical disk space without physically zeroing the contents. This enables SQL Server to consume the physical disk space quickly. All versions of SQL Server use the database allocation structures to determine the valid pages in the database; every time that a new data page is allocated, it is formatted and written to stable media.

SQL Server 2000 creates and expands database log and data files by stamping the new section of the file by using all zero values. A SQL Server 2005 instance with incorrect account privileges will revert to SQL Server 2000 behavior. The algorithm used by SQL Server is more aggressive than the NTFS zero initialization (DeviceIoControl, FSCTL_SET_ZERO_DATA), thereby elevating the NTFS file lock behavior and enabling concurrent access to other sections of the file. However, zero initializing is limited by physical I/O capabilities and can be a lengthy operation.

SQL Server 2005 uses instant file initialization only for data files. Instant file initialization removes the zero stamping during the creation or growth of the data file. This means that SQL Server 2005 can create very large data files in seconds.

The following rules apply to SQL Server 2005 and instant file initialization.

The operating system and file system must support instant file initialization.

The SQL Server startup account must possess the `SE_MANAGE_VOLUME_NAME` privilege. This privilege is required to

successfully run `SetFileValidData`.

The file must be a SQL Server database data file. SQL Server transaction log files are not eligible for instant file initialization.

If trace flag `-T1806` is enabled, SQL Server 2005 behavior reverts to SQL Server 2000 behavior.

`SetFileValidData` does allow for fast allocation of the physical disk space. High-level permissions could enable data that already exists on the physical disk to be seen, but only internally, during a SQL Server read operation. Because SQL Server knows which pages have been allocated, this data is not exposed to a user or administrator.

To guarantee transaction log integrity, SQL Server must zero initialize the transaction log files. However, for data files SQL Server formats the data pages as they are allocated so the existing data does not pose a problem for database data files.

> **Note:** To guarantee the physical data file space acquisition during data file creation or expansion, on a thin provisioned subsystem, use trace flag –T1806.

Trace flag –T1806 provides backward compatibility to zero initialize database data files without using instant file initialization of files. You may also remove the `SE_MANAGE_VOLUME_NAME` privilege to force SQL Server to use zero file initialization. For more information on the `SE_MANAGE_VOLUME_NAME` privilege, see Database File Initialization in SQL Server 2005 Books Online.

> **Note**: Zero file initialization does not protect against previously stored data discovery. To fully prevent discovery of previous data, the physical media must have a Department Of Defense (DOD)-level series of write. This is generally seven unique write patterns. Zero file initialization only performs a single, all-zero write to the data pages.
>
> For more information on previously stored data security, see HYPERLINK "http://www.microsoft.com/athome/moredone/protectpurgepersonalfiles.mspx" Protect and purge your personal files (**HYPERLINK "http://www.microsoft.com/athome/moredone/protectpurgepersonalfiles.mspx"** http://www.microsoft.com/athome/moredone/protectpurgepersonalfiles.mspx).
>
> For more information about DOD-5012.2 STD, see HYPERLINK "http://www.dtic.mil/whs/directives/corres/pdf/50152std_061902/p50152s.pdf" Design Criteria For Electronics Record Management Software Applications (HYPERLINK "http://www.dtic.mil/whs/directives/corres/pdf/50152std_061902/p50152s.pdf" http://www.dtic.mil/whs/directives/corres/pdf/50152std_061902/p50152s.pdf).

I/O affinity and snapshot backups

I/O affinity dedicates special, hidden schedulers to performing core buffer pool I/O and log writer activities. The I/O affinity configuration option was introduced in SQL Server 2000 SP1. The HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;298402" INF: Understanding How to Set the SQL Server I/O Affinity Option (HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;298402" http://support.microsoft.com/default.aspx?scid=kb;en-us;298402) Microsoft Knowledge Base article outlines the I/O affinity implementation.

Vendors can use Virtual Device (VDI)-based snapshot backups to perform actions such as splitting a mirror. To accomplish this, SQL Server must guarantee point-in-time data stability and avoid torn writes by freezing all new I/O operations and completing all outstanding I/Os for the database. The Windows Volume Shadow Copy Service (VSS) backup is one such VDI application.

For more information on VDI snapshot backups, see Snapshot Backups in SQL Server Books Online.

For more information on VDI Specifications, see HYPERLINK "http://www.microsoft.com/downloads/details.aspx?FamilyID=416f8a51-65a3-4e8e-a4c8-adfe15e850fc&DisplayLang=en" SQL Server 2005 Virtual Backup Device Interface (VDI) Specification (HYPERLINK http://www.microsoft.com/downloads/details.aspx?FamilyID=416f8a51-65a3-4e8e-a4c8-adfe15e850fc&DisplayLang=en http://www.microsoft.com/downloads/details.aspx?FamilyID=416f8a51-65a3-4e8e-a4c8-adfe15e850fc&DisplayLang=en).

The SQL Server 2000 design does not account for frozen I/O in combination with I/O affinity. A single database snapshot backup freezes I/O requests (reads and writes) for all databases. This makes I/O affinity and snapshot backups an unlikely combination choice for SQL Server 2000.

SQL Server 2005 supports I/O affinity as outlined in the article mentioned earlier (HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;298402" INF: Understanding How to Set the SQL Server I/O Affinity Option). It corrects the condition which could freeze all I/O requests when I/O affinity is enabled.

SQL Server I/O affinity is a very specialized, high-end server configuration option. It should only be used after significant testing indicates that it will result in performance improvements. A successful I/O affinity implementation increases overall throughput. The I/O affinity schedulers maintain reasonable CPU usage levels and effectively allow other schedulers access to increased CPU resources.

Locked memory pages

Both 32-bit and 64-bit versions of SQL Server use the AWE API set to

lock pages in memory as `VirtualLock` is not guaranteed to keep all locked pages in memory like `AllocateUserPhysicalPages`. This can sometimes be confusing because the AWE **sp_configure** settings exist but are not used on 64-bit SQL Server. Instead, the lock pages privilege determines when to use the AWE API set. The Windows "Lock Pages In Memory" privilege is necessary for SQL Server to make AWE allocations.

Lock Pages In Memory can have a positive affect on I/O performance and can prevent the trimming of the working set of SQL Server.

> **Warning:** Do not use AWE or locked pages without testing. Forcing strict, physical memory usage can result in undesired physical memory pressure on the system. System-level or hardware components may not perform well when physical memory pressure is present. Use this option with caution.

During an I/O operation, the memory for the I/O should not cause page faults. Therefore, if the memory is not already locked, it will be. Because most applications do not lock I/O memory, the operating system transitions the memory to a locked state. When the I/O finishes, the memory is transitioned back to an unlocked state.

SQL Server 2005 64-bit Enterprise Edition detects if the Lock Pages In Memory privilege is present and establishes a locked pages data cache. SQL Server 32-bit installations require enabling AWE memory options to establish the locked memory behavior. This avoids the lock and unlock transition during I/O, thereby providing improved I/O performance. To disable this behavior, remove the privilege or, for 64-bit installations, enable startup trace flag `-T835`.

> **Important:** SQL Server always tries to avoid paging by releasing memory back to the system when it is possible. However, certain conditions can make this difficult. When pages are locked they cannot be paged. We recommend careful consideration of locked page usage.

It is not always possible for SQL Server to avoid paging operations. Locked pages can also be used to avoid paging as a troubleshooting technique if you have reason to believe damage to memory may have occurred during a paging operation.

Idle server

SQL Server 2005 can provide an idle SQL Server process (sqlservr.exe) similar to the suspend/resume operations that are provided by the operating system. SQL Server 2005 introduces a per-node, system task called Resource Monitor. Resource Monitor's primary responsibility is to watch core memory levels and then trigger cache adjustments accordingly. When the SQL Server 2005 Resource Monitor detects that there are no user-initiated requests remaining to process, it can enter the idle state. When a new user request arrives or a critical event must run, SQL Server wakes and handles the activities. The following table

outlines some of the key event/request types and associated actions as they relate to the idle SQL Server process.

Event/Request Type	Description
User Request	A user request includes those actions initiated by a client application that require SQL Server to perform a service. The following are common examples of user requests. Batch request SQL RPC request DTC request Connection request Disconnect request Attention request These actions are also called *external requests*. Active user requests prevent SQL Server from entering an idle state. When SQL Server is in an idle state, a user request wakes the SQL Server process.
Internal Tasks	An internal request includes those actions that a user cannot issue a specific client request to trigger or control. For example: Lazy writer Log writer Automatic checkpointing Lock monitor Internal tasks do not prevent SQL Server from entering an idle state. Only critical internal tasks can wake the SQL Server from an idle state.

Critical Event	Some SKUs of SQL Server respond to events such as memory notifications in order to wake the SQL Server process from an idle state and honor the event.

There are some basic rules the SQL Server process uses to determine whether it can enter an idle state.

The SQL Server idle state is not entered until:

No user requests have been active in 15 minutes.

The instance is not participating in database mirroring.

Service Broker is in an idle state.

SQL Server wakes in response to:

Memory pressure (except on SQL Server Express).

A critical internal task or event.

An external request such as a Tabular Data Stream (TDS) packet arrival.

The idle activity can change the way SQL Server interacts with system. The following table outlines specific behavior related to the individual SKUs.

SKU	Behavior
SQL Server Express Service	By default can declare the server idle. Triggers the operating system to aggressively trim the SQL Server working set memory using API `SetProcessWorkingSetSize(…, -1, -1)` Does not wake in response to operating system memory pressure notifications. Tries to enter an idle state immediately after service startup.
SQL Express Individual User Instance	By default can declare the server idle. May be configured to allow for aggressive working set trimming. Wakes in response to operating system memory pressure notifications.

Workgroup	By default can declare the server idle. May be configured to allow for aggressive working set trimming. Wakes in response to operating system memory pressure notifications
Standard **Enterprise** **Developer**	Idle server behavior requires a trace flag. Wakes in response to operating system memory notification.

SQL Server always tries to avoid using the page file whenever possible. However, it is not always possible to avoid all page file activity. An idle SQL Server process may introduce aggressive use of the page file and increased I/O path usage even though SQL Server tries to avoid these. The paging activities require that memory regions be stored and retrieved on disk. This opens the possibility of memory corruption if the subsystem is not handling the paging activity correctly.

> **Note:** The Windows operating systems use storage and read-ahead design techniques for the page file that are similar to that which SQL Server uses for data and log files. This means that an idle SQL Server can experience I/O patterns during a paging operation similar to that of its own data file reads and writes. To guarantee data integrity, the page file should uphold I/O specifications suited for SQL Server database and log files.

SQL Server idle server activity can be controlled with the following trace flags.

Trace Flag	Description
8009	Enable idle server actions
8010	Disable idle server actions

Database mirroring (DBM)

SQL Server 2005 SP1 introduces the database mirroring (DBM) feature set. For more information about database mirroring solutions, see Database Mirroring in SQL Server 2005 Books Online.

Database mirroring is not specifically targeted as a remote mirroring solution. However, database mirroring addresses the necessary properties required to maintain the primary and mirror relationship, guarantee data integrity, and allow for both failover and failback to occur as outlined in HYPERLINK \l "_Remote_Mirroring" Remote Mirroring earlier in this paper.

Database mirroring uses CRC checks to guarantee data transmissions between the primary and mirror to maintain the correct data integrity. SQL Server 2005, for new databases, provides checksum capabilities for data pages and log blocks to strengthen data integrity maintenance. To enhance your 'Always On' solution, we recommend using the CRC transmission checks in combination with the checksum database capabilities to provide the highest high level of protection against data corruption.

Multiple instance access to read-only databases

SQL Server 2005 introduces Scalable Shared Database support (SSD), enabling multiple SQL Server instances to use the same read-only database from a read-only volume. The setup details and other detailed information about SSD are outlined in the SQL Server 2005 Books Online Web Refresh and in the following Microsoft Knowledge Base article: HYPERLINK "http://support.microsoft.com/?kbid=910378" Scalable Shared Databases are supported by SQL Server 2005 (HYPERLINK "http://support.microsoft.com/?kbid=910378" http://support.microsoft.com/?kbid=910378).

SSD provides various scale out possibilities, including but not limited to the following:

Multiple server access

Separate server resources

Separate **tempdb** resources

Note: Multiple server, database file access is never supported for SQL Server databases in read/write mode. SQL Server SSD is never supported against writable database files.

Some I/O subsystems support features such as volume-level, copy-on-write (COW) snapshots. These I/O subsystem solutions monitor write operations on the primary (read/write) volume and save the initial data to the volume snapshot. The volume snapshot is presented as a read-only copy when mounted. The read-only, snapshot volume is a supported configuration for SQL Server SSD databases. Such implementations can provide a powerful tool for accessing live production data, read only, from many servers.

Some subsystems use distributed locking mechanisms instead of read-only volume snapshot capabilities. This allows multiple servers to access the same read/write volume. The distributed locking environments are very powerful but do not present a point-in-time image of the data to secondary servers. The live data is presented to all servers connected to the volume. SQL Server SSD is not supported against the writable database files. The SQL Server buffer pool cannot maintain point-in-time synchronization on secondary servers. This

would lead to various problems because of the resulting dirty read activities.

Ramp up of local cache

The Enterprise Edition of SQL Server 2005 tries to ramp up a node's local data cache during the node initialization phase. The initial memory growth committal of a node is considered to be the ramp-up period. During the ramp-up period, whenever a single page read is requested, the request is turned into an eight-page request (64 KB) by the buffer pool. The ramp-up helps reduce waits for subsequent single page reads after a restart. This enables the data cache to be populated quicker and SQL Server to return to cached behavior quicker.

A non-NUMA computer is considered to have a single-node implementation. On larger memory installations, this can be a significant advantage because it enables quicker data cache repopulation. Each node is assigned and monitored by a separate Resource Monitor task. SQL Server is aware of the different nodes and different memory requirements that may exist within those nodes. This includes the buffer pool. This makes SQL Server fully NUMA aware.

Encrypted file systems (EFS)

SQL Server databases can be stored in NTFS encrypted files. However, use this feature with caution as encryption actions disable asynchronous I/O capabilities and lead to performance issues. When performing I/O on an EFS-enabled file, the SQL Server scheduler becomes stalled until the I/O request completes. Actions such as SQL Server read ahead become disabled for EFS files. We recommend using the built-in SQL Server 2005 encryption capabilities instead of EFS when possible.

Use of EFS for SQL Server should be limited to physical security situations. Use it in laptop installations or other installations where physical data security could be at risk.

If EFS must be deployed in server environment consider the following.

Use a dedicated SQL Server instance.

Test throughput limitations well.

Test with and without the I/O affinity mask. Using I/O affinity may provide a pseudo-async capability to SQL Server.

DiskPar.exe

The diskpar utility provides alignment capabilities to prevent misalignment performance problems. Systems running SQL Server should properly align on boundaries to help optimize performance. I/O subsystem manufactures have established recommendations for proper alignment in a SQL Server environment.

This is the same recommendation as for Microsoft Exchange Server. The following is an except from the Microsoft Exchange documentation and is applicable to SQL Server.

> "Even though some storage obfuscates sectors & tracks, using diskpar will still help by preventing misalignment in cache. If the disk is not aligned, every Nth (usually 8th) read or write crosses a boundary, and the physical disk must perform two operations.
>
> At the beginning of all disks is a section reserved for the master boot record (MBR) consuming 63 sectors. This means that your partition will start on the 64th sector, misaligning the entire partition. Most vendors suggest a 64-sector starting offset.
>
> Check with your particular vendor before standardizing this setting on a particular storage array."

Always On high-availability data storage

SQL Server 2005 introduces the Always On storage review program for high-availability solutions. Various manufacturers have reviewed their solutions against Microsoft SQL Server requirements and have published detailed white papers about how their solutions can be used with SQL Server to maintain the Always On goal. For more information, see HYPERLINK "http://www.microsoft.com/sql/AlwaysOn" SQL Server Always On Storage Solution Review Program (HYPERLINK "http://www.microsoft.com/sql/AlwaysOn" www.microsoft.com/sql/AlwaysOn).

SQLIOSim

SQLIOSim replaces SQLIOStress. It is used to test SQL Server I/O patterns without requiring that SQL Server be installed. *The tests do not use actual SQL Server database and log files but simulate them instead.* This utility greatly extends testing capabilities by enabling control over memory footprint, multiple files per database, shrink and grow actions, files larger than 4 GB and other options.

We recommend using SQLIOSim to test the system before you install SQL Server. This will help you to improve your data safety.

> **Note**: If SQL Server is reporting corruption or other I/O subsystem error conditions, back up your data and then run the SQLIOSim testing utility to test the I/O subsystem in addition to running other hardware check utilities provided by your hardware manufacture.
>
> **Important:** SQLIOSim and SQLIOStress are also used by the Microsoft Hardware Compatibility Labs and by various vendors to make sure that the I/O subsystem is SQL Server I/O pattern compliant. If SQLIOSim or SQLIOStress return errors, this clearly indicates that the I/O subsystem is not performing at an HCL-compliant level. This could lead to severe data damage or loss.

Conclusion

For more information:

HYPERLINK "http://www.microsoft.com/technet/prodtechnol/sql/default.mspx" http://www.microsoft.com/technet/prodtechnol/sql/default.mspx

Did this paper help you? Please give us your feedback. On a scale of 1 (poor) to 5 (excellent), HYPERLINK "mailto:sqlfback@microsoft.com?subject=White%20Paper%20Feedback:%20Microsoft%20SQL%20Server%20I/O%20Basics:%20Chapter%202"how would you rate this paper?

References

There are many aspects to consider when you are setting up the I/O subsystem. This section provides a list of documents that you might consider reading.

For updated I/O details, you can also visit the HYPERLINK "http://www.microsoft.com/sql/support" http://www.microsoft.com/sql/support Web page.

SQL Server Always Storage Solution Review Program

HYPERLINK "http://www.microsoft.com/sql/AlwaysOn" http://www.microsoft.com/sql/AlwaysOn

Certification Policy

KB913945- HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;EN-US;913945" Microsoft does not certify that third-party products will work with Microsoft SQL Server

KB841696 - HYPERLINK "http://support.microsoft.com/kb/841696/en-us" Overview of the Microsoft third-party storage software solutions support policy

KB231619 - HYPERLINK "support.microsoft.com/kb/q231619/" How to use the SQLIOStress utility to stress a disk subsystem such as SQL Server

Fundamentals and Requirements

White paper- HYPERLINK "http://www.microsoft.com/technet/prodtechnol/sql/2000/maintain/sqlIObasics.mspx" SQL Server 2000 I/O Basics (applies to SQL Server versions 7.0, 2000, and 2005)

KB230785 - HYPERLINK "http://support.microsoft.com/kb/230785/en-us" SQL Server 7.0, SQL Server 2000 and SQL Server 2005 logging and data storage algorithms extend data reliability

KB917047 - HYPERLINK "http://support.microsoft.com/?kbid=917047" Microsoft SQL Server I/O subsystem requirements for the tempdb database

KB231347 - HYPERLINK "http://support.microsoft.com/kb/231347/en-us" SQL Server databases not supported on compressed volumes (except 2005 read only files)

Subsystems

KB917043 - HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;917043&sd=rss&spid=2852" Key factors to consider when evaluating third-party file cache systems with SQL Server

KB234656- HYPERLINK "http://support.microsoft.com/default.aspx?scid=KB;%5bLN%5d;234656" Using disk drive caching with SQL Server

KB46091- HYPERLINK "http://support.microsoft.com/kb/46091/en-us" Using hard disk controller caching with SQL Server

KB86903 - HYPERLINK "http://support.microsoft.com/kb/86903/en-us" Description of caching disk controls in SQL Server

KB304261- HYPERLINK "http://support.microsoft.com/default.aspx?scid=KB;%5bLN%5d;304261" Description of support for network database files in SQL Server

KB910716 (in progress) - HYPERLINK "http://support.microsoft.com/kb/910716/" Support for third-party Remote Mirroring solutions used with SQL Server 2000 and 2005

KB833770 - HYPERLINK "HYPERLINK%20%22http:/support.microsoft.com/kb/833770/en-us" Support for SQL Server 2000 on iSCSI technology components (applies to SQL Server 2005)

Design and Configuration

White paper - HYPERLINK "http://download.microsoft.com/download/4/f/8/4f8f2dc9-a9a7-4b68-98cb-163482c95e0b/PhysDBStor.doc" Physical Database Layout and Design

KB298402 - HYPERLINK "http://support.microsoft.com/kb/298402/en-us" Understanding How to Set the SQL Server I/O Affinity Option

KB78363 - HYPERLINK "http://support.microsoft.com/kb/78363/en-us" When Dirty Cache Pages are Flushed to Disk

White paper - HYPERLINK "http://www.microsoft.com/technet/prodtechnol/sql/2005/dbmirror.mspx" Database Mirroring in SQL Server 2005

White paper - HYPERLINK "http://download.microsoft.com/download/4/f/8/4f8f2dc9-a9a7-4b68-98cb-163482c95e0b/DBM_Best_Pract.doc"Database Mirroring Best Practices and Performance Considerations

KB910378 - HYPERLINK "http://support.microsoft.com/Default.aspx?kbid=910378" Scalable shared database are supported by SQL Server 2005

MSDN article - HYPERLINK "http://msdn2.microsoft.com/en-us/library/ms190257(SQL.90).aspx"Read-Only Filegroups

KB156932 - HYPERLINK "http://support.microsoft.com/default.aspx?scid=kb;en-us;156932" Asynchronous Disk I/O Appears as Synchronous on Windows NT, Windows 2000, and Windows XP

Diagnostics

KB826433 - HYPERLINK "http://support.microsoft.com/default.aspx?scid=KB;%5bLN%5d;826433" Additional SQL Server Diagnostics Added to Detect Unreported I/O Problems

KB897284 - HYPERLINK "http://support.microsoft.com/kb/897284/en-us" SQL Server 2000 SP4 diagnostics help detect stalled and stuck I/O operations (applies to SQL Server 2005)

KB828339 - HYPERLINK "http://support.microsoft.com/default.aspx?

scid=KB;%5bLN%5d;828339" Error message 823 may indicate hardware problems or system problems in SQL Server

KB167711 - HYPERLINK "http://support.microsoft.com/kb/167711/en-us" Understanding Bufwait and Writelog Timeout Messages

KB815436 - HYPERLINK "http://support.microsoft.com/kb/815436/en-us" Use Trace Flag 3505 to Control SQL Server Checkpoint Behavior

KB906121 - HYPERLINK "http://support.microsoft.com/kb/906121/en-us" Checkpoint resumes behavior that it exhibited before you installed SQL Server 2000 SP3 when you enable trace flag 828

WebCast- HYPERLINK "http://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?EventID=1032290646&EventCategory=5&culture=en-US&CountryCode=US" Data Recovery in SQL Server 2005

Known Issues

KB909369 - HYPERLINK "http://support.microsoft.com/kb/909369/en-us" Automatic checkpoints on some SQL Server 2000 databases do not run as expected

KB315447 - HYPERLINK "http://support.microsoft.com/kb/315447/en-us" SQL Server 2000 may be more aggressive with Lazy Writers than SQL Server 7.0

KB818767 - HYPERLINK "http://support.microsoft.com/kb/818767/en-us" Improved CPU Usage for Database Logging When Transaction Log Stalls Occur

KB815056 - HYPERLINK "http://support.microsoft.com/kb/815056/en-us" You receive an "Error: 17883" error message when the checkpoint process executes

KB915385 HYPERLINK "http://support.microsoft.com/?kbid=915385" A snapshot-based database backup restore process may fail, and you may receive an error message in SQL Server 2005

HYPERLINK "http://www.microsoft.com/sql/support" Support Assistance (http://www.microsoft.com/sql/support)

Utilities

Download - HYPERLINK "http://www.microsoft.com/downloads/details.aspx?FamilyID=9a8b005b-84e4-4f24-8d65-cb53442d9e19&DisplayLang=en" SQLIO Disk Subsystem Benchmark Tool

Download - HYPERLINK "http://support.microsoft.com/kb/231619/en-us" SQLIOStress utility to stress disk subsystem (applies to SQL Server 7.0, 2000, and 2005 - replaced with SQLIOSim)

PAGE ii

PAGE ii

PAGE iii

PAGE iii

PAGE 54

PAGE 54

PAGE 55

PAGE 55