High Availability with SQL Server 2008

SQL Server Technical Article

Writer: Paul S. Randal (SQLskills.com)

Technical Reviewers: Sanjay Mishra, Michael Thomassy, Haydn Richardson, Gopal Ashok, Kimberly L. Tripp (SQLskills.com), Glenn Berry (NewsGator Technologies)

Published: September 2009

Applies to: SQL Server 2008

Summary:

This white paper describes the technologies available in SQL Server 2008 that can be used as part of a high-availability strategy to protect critical data. As well as describing the technologies in detail, the white paper also discusses the various causes of downtime and data loss, and how to evaluate and balance requirements and limitations when planning a high-availability strategy involving SQL Server 2008.

This white paper is targeted at architects, IT pros, and database administrators (DBAs) tasked with implementing a high-availability strategy. It assumes the reader is familiar with Windows and SQL Server and has at least a rudimentary knowledge of database concepts such as transactions.

Introduction

Today many companies require some or all of their critical data to be highly-available. For example, a company requiring "24x7" availability is an online merchant, whose product databases and online sales applications must be available at all times; otherwise sales (and revenue) are lost. Another example is a hospital, where computerized patient records must be available at all times or a human life could be lost.

In a perfect world, this critical data would remain available and nothing would threaten its availability. In the real world, however, there are numerous problems that can cause data to become unavailable. If high availability of the data is required, a proactive strategy must be formulated to mitigate the threats to availability—commonly called a "high-availability strategy".

Such strategies always call for the implementation of multiple technologies that help maintain data availability—there is no single high-availability technology that can meet all requirements. The Microsoft® SQL Server® 2008 data management system includes a variety of technologies that can be used to increase and/or maintain high availability of critical data. This white paper will introduce these technologies and describe when and how they should be used.

It should be noted here that high availability is not the same as disaster recovery, although the two terms are often (erroneously) interchanged. High availability is about putting a set of technologies into place *before* a failure occurs to prevent the failure from affecting the availability of data. Disaster recovery is about taking action *after* a failure has occurred to recover any lost data and make the data available again.

Before describing SQL Server 2008 high-availability technologies and their uses, the white paper will discuss the causes of downtime and data loss, to allow the usefulness and applicability of the technologies to be shown. Furthermore, the white paper will also present guidelines on how to approach planning for high availability—including what requirements to gather and what limitations to be aware of before considering any technologies.

Finally, after the technologies have been described, the causes of downtime and data loss will be revisited with

a list of which technologies can be used to mitigate each cause.

Many of the technologies discussed are only available in SQL Server 2008 Enterprise, while others have limitations in SQL Server 2008 Standard (or lower). A summary table of edition support for the features discussed in the white paper is included in the final section. Furthermore, the SQL Server Books Online topic "Features Supported by the Editions of SQL Server 2008" (http://msdn.microsoft.com/en-us/library/cc645993.aspx [http://msdn.microsoft.com/en-us/library/cc645993.aspx]) has a comprehensive list.

Causes of Downtime and Data Loss

The main point of a high-availability strategy is to keep the critical data as available as possible in the event of a failure or disaster. Failures, by their very nature, are not planned events and so it is hard to predict when one will occur and what form it will take. There are also times when planned events may occur that affect data availability if preventative steps have not been taken. This section of the white paper will discuss the various events that may occur that can cause downtime and data loss, but it will not discuss the mitigating technologies.

Planned Downtime

There are three planned activities that can cause downtime—performing database maintenance, performing batch operations, and performing an upgrade. Neither of these activities results in data loss.

Database maintenance can cause downtime if an operation is performed that needs to place a lock on a table for an extended period of time. Examples of such operations include:

- Creating or rebuilding a nonclustered index (can prevent table modifications)
- Creating, dropping, or rebuilding a clustered index (can prevent table reads and modifications)

Performing batch operations can cause downtime through blocking locks. For example, consider a table that holds the year-to-date sales for a company. At the end of each month, the data from the oldest month must be deleted. If the number of rows being deleted is large enough, a blocking lock may be required, which prevents updates to the table while the delete operation is being performed. A similar scenario exists where data is being loaded into a table from another source.

Performing an upgrade always incurs some amount of downtime, because there comes a point where the application must disconnect from the database that is being upgraded. Even with a hot standby technology (such as synchronous database mirroring), there is (at best) a very short time between the application disconnecting and then connecting to the redundant copy of the database.

In both cases, planned downtime can be minimized using the technologies in SQL Server 2008.

Unplanned Downtime and Data Loss

An even larger number of failures exist that can cause downtime and data loss. These can be grouped into several broad categories:

- Data center failure: This category of failures takes the entire data center offline, rendering any local redundant copies of the data useless. Examples include natural disasters, fire, power loss, or failed network connectivity.
- Server failure: This category of failures takes the server hosting one or more SQL Server instances offline. Examples include failed power supply, failed CPU, failed memory, or operating system crashes.
- I/O subsystem failure: This category of failures involves the hardware used to physically store the data and thus directly affects the data itself—usually causing data loss (which then may lead to downtime to perform disaster recovery). Examples include a drive failure, a RAID controller failure, or an I/O subsystem software bug causing corruption.
- Human error: This category of failures involves someone (a DBA, a regular user, or an application programmer introducing an application software bug) making a mistake that damages data, usually causing data loss (and then potentially downtime to recover the data). Examples include dropping a table, deleting or updating data in a table without specifying a predicate, setting a database offline, or shutting down a SQL Server instance. Human errors are usually accidental, but they can also be malicious.

In all cases, data loss can be prevented and downtime can be minimized using the technologies in SQL Server 2008.

Planning a High-Availability Strategy

A successful high-availability strategy cannot be planned solely from the technical standpoint, because the costs and risks to the business from downtime and/or data loss must be understood. Similarly, a strategy cannot be driven solely from business requirements without an understanding of the technical implications of those requirements.

The first answer from many people when planning a high-availability strategy is something like "implement failover clustering!" without any consideration of what the strategy is trying to achieve. Although failover clustering is an excellent technology, it is not appropriate in all situations, so it is important to pick the right technologies rather than just the first one that comes to mind. (Again, note that a successful strategy always includes multiple technologies.)

Being able to pick the right technologies means understanding not only the characteristics of the technologies but also the prioritized list of requirements, taking into account any limitations that exist.

Requirements

Requirements are an important part of the design and testing phases. As part of the overall design process, requirements are identified. Next, the design must be validated against the requirements prior to implementation, and finally, they must be used to test the design after it is implemented.

The aim of the requirements gathering process is to generate a prioritized list of what data needs to be protected and to what level. For example, the process should take into account system components, the amount of time and data the organization can afford to lose, and performance.

It is important to consider the "application ecosystem"— the data components of the system that must be protected. This may be as simple as a few rows in a table or as complicated as multiple databases on different SQL Server 2008 instances, plus appropriate SQL Server Agent jobs, security settings, stored-procedures, and more. The more complicated the application ecosystem, the more difficult it is to ensure everything is made highly available.

Of course, it is likely that the overall high-availability strategy will encompass protecting multiple resources with different requirements and against different causes of downtime and data loss.

The two main requirements around high-availability are commonly known as RTO and RPO. RTO stands for Recovery Time Objective and is the maximum allowable downtime when a failure occurs. RPO stands for Recovery Point Objective and is the maximum allowable data-loss when a failure occurs. Apart from specifying a number, it is also necessary to contextualize the number. For example, when specifying that a database must be available 99.99% of the time, is that 99.99% of 24x7 or is there an allowable maintenance window?

A requirement that is often overlooked is workload performance. Some high-availability technologies can affect workload performance when implemented (either outright or when configured incorrectly). Also, workload performance after a failover must be considered—should the workload continue with the same throughput as before, or is some temporary degradation acceptable?

Some examples of requirements are:

- Database X must be available as close to 24x7 as possible and no data loss can be tolerated. Database X also relies on stored procedures in the master database and must be hosted on a SQL Server 2008 instance on a server in security domain Y.
- Tables A, B, and C in database Z must be available from 8 A.M. to 6 P.M. on weekdays, no data loss can be tolerated, and they must all be available together. After a failover, workload performance cannot drop.

Limitations

Limitation analysis is crucial to prevent wasted time (and money) from designing a strategy that cannot be implemented (for example, the strategy involves adding a data center, but the total budget is US$10,000 per year). Limitations are not just in terms of budget (although that is usually the overriding limitation)—there are

other nontechnical limitations, and a host of technical limitations to consider.

The nontechnical limitations include:

- Power (for more servers, disks, and associated air conditioning)
- Space (for more servers and ancillary equipment)
- Air conditioning (to cope with all the extra heat output from added equipment)
- Manpower (to install and maintain any added systems and equipment)
- Time to implement (the more complex the design, the longer it takes to implement)
- Politics and/or management issues (if multiple teams are involved)

The technical limitations can more directly affect which SQL Server 2008 technologies you can choose to incorporate in the high-availability strategy. Some important questions to ask are listed here, and the ramifications of the answers are explained in more detail throughout the rest of the white paper.

What recovery model is being used currently? Database mirroring, log shipping, and the ability to take transaction log backups are not available if the simple recovery model is used. Switching out of the simple recovery model requires that transaction log management be performed to prevent the transaction log growing out of control.

What is the transaction log generation rate of the workload? This impacts the management of the transaction log itself, especially because some technologies require the full recovery model (for example, database mirroring). If the recovery model has to change, many things may be affected, such as the size of transaction log backups or the ability to perform faster batch processes and index maintenance operations.

How large are the average transactions? Large (and usually long-running) transactions can significantly affect the ability to quickly bring a redundant copy of a database online after a failover, because any uncommitted transactions must be rolled back to make the database transactionally consistent. This may prevent the RTO from being achieved. Also, if synchronous database mirroring is being considered, the network latency and its effect on transaction response time should be taken into account.

What is the network bandwidth and latency to the remote data center? Poor bandwidth and/or high latency affect the ability to implement any technology that copies data, transaction log content, or backups between data centers.

Can the application be altered? With any redundancy technology, the application must reconnect to the server (or maybe another server) after a failover occurs, and potentially restart a failed transaction. These are much easier to accomplish if the application can be altered.

Is the FILESTREAM feature required in the application? The FILESTREAM feature cannot be used in conjunction with database mirroring or database snapshots in SQL Server 2008. Also, depending on how FILESTREAM is used, it may cause problems with backups, log shipping, and transactional replication. This is explained in more detail in the white paper "FILESTREAM Storage in SQL Server 2008" (available at http://msdn.microsoft.com/en-us/library/cc949109.aspx [http://msdn.microsoft.com/en-us/library/cc949109.aspx]).

Are there any hardware or software limitations? Hardware limitations (in terms of CPU, memory, I/O subsystem, or disk space) can affect database mirroring, multi-instance failover clustering, RAID level, and workload throughput. The available edition of SQL Server 2008 affects which high-availability technologies are able to be used (as mentioned in the Introduction).

After all the requirements and limitations are known, realistic analysis is needed to ensure that the original requirements can be met or altered to meet identified limitations. Compromises may involve relaxing a requirement or removing a limitation, or simply designing a strategy that meets as many of the requirements as possible with defined and documented limitations. Management should also be made aware of these limitations and approve of the compromises – otherwise additional budget, time, or analysis might be needed to reach higher levels of availability.

Whatever form the compromise takes, it is essential that all interested parties agree on the compromise. And again, it is also essential that management is aware of which requirements cannot be met and why *before* a failure occurs, so that no one is surprised.

Technology Evaluation

After the post-compromise requirements are known then, and only then, is the time to evaluate the various technologies. There is no point evaluating and picking technologies based on a flawed set of requirements and

then having to repeat the process after the requirements are better understood.

Similarly, success does not come from picking an unsuitable technology and then trying to make it perform a function for which it was not designed. For example:

- Log shipping is unsuitable for a design that requires zero data loss.
- Database mirroring is unsuitable for a design that requires multiple databases to fail over simultaneously.

When you evaluate technologies, it is important to consider all aspects of the technologies, including:

- The monetary cost of implementing the technology.
- The complexity of implementing, configuring, and managing the technology.
- The technology's impact on workload performance (if any).
- The data loss exposure if the technology is used.
- The downtime potential if the technology is used.

SQL Server 2008 High-Availability Technologies

SQL Server 2008 has a large number of technologies to aid in maintaining high availability. The easiest way to describe these is to start with logging and recovery, backups, and backup strategy—the building blocks of any high-availability strategy. This provides the basis to then explain the other technologies that are applicable if only a single instance of SQL Server 2008 can be used, and then the multi-instance technologies, which people usually think of as the high-availability technologies in SQL Server.

Logging and Recovery

After the post-compromise requirements are known then, and only then, is the time to evaluate the various technologies. There is no point evaluating and picking technologies based on a flawed set of requirements and then having to repeat the process after the requirements are better understood.

The fundamental mechanisms that allow SQL Server 2008 (and all other relational database management systems in the industry) to keep data transactionally consistent are logging and recovery.

In this section these two mechanisms will be briefly described. They underpin nearly all the high-availability technologies in SQL Server 2008. For a much more in-depth description of these mechanisms, and a description of the architecture and behavior of the transaction log itself, see the TechNet Magazine article "Understanding Logging and Recovery in SQL Server" (available at http://technet.microsoft.com/en-us/magazine/2009.02.logging.aspx [http://technet.microsoft.com/en-us/magazine/2009.02.logging.aspx]).

All changes to a database are logged as they are made. Logging a change means describing exactly what changed in the database and how it was changed—in a structure called a log record. Log records are stored in the transaction log for each database and must remain available until they are no longer required by SQL Server. Some common reasons for log records being required include:

- A transaction log backup needs to be taken.
- A full or differential backup requires the transaction log.
- Database mirroring has not yet sent the log records to the mirror database.
- Transactional replication has not harvested the committed log records.
- Change data capture has not harvested the committed log records.
- Uncommitted transactions exist which may be rolled back.

The primary reason that logging takes place is to allow crash recovery to happen (sometimes called *restart recovery* or simply *recovery*).

Recovery is the mechanism by which a database is made transactionally consistent. It occurs when a database is brought online after being shut down with active transactions running (for example, after a SQL Server 2008 instance fails over to another node in a failover cluster). It is also required as part of restoring backups (whether manually or as part of log shipping), running database mirroring, and creating a database snapshot.

Recovery has two phases—redo and undo. The redo phase ensures that all committed (that is, completed) transactions are wholly reflected in the database. Redo is usually faster than undo (but of course it is very

variable). The undo phase ensures that all uncommitted (that is, active but not completed) transactions are not reflected in the database in any way whatsoever. In other words, the database is made transactionally consistent.

There is a special case of undo—when a transaction is rolled back during the course of normal operations. In that case, all changes made by the transaction are reversed. This is commonly known as *rolling back* or *aborting* a transaction.

A database cannot be made available until recovery has completed, with two exceptions. In SQL Server 2008 Enterprise, the database is made available after the redo phase has completed during regular crash recovery or as part of a database mirroring failover. In these two cases, the database comes online faster and downtime is reduced. This feature is called *fast recovery*.

Backup, Restore, and Related Technologies

Any high-availability strategy must include a comprehensive backup strategy. Even if the high-availability strategy includes technologies to provide redundant copies of the databases being protected, there are several reasons why backups should also exist:

- The failure may affect the redundant copies of the databases as well as the primary copies. In this case, without backups to allow restoration, the data may be completely lost.
- It may be possible to restore a small portion of the databases, which may be more palatable than failing over. Furthermore, it may be possible to restore that small portion of the database without affecting the portion being used by the application—effectively recovering from the failure with zero downtime.

One mistake that many people make is to plan a backup strategy without carefully considering the consequences of having to perform restore operations. It may seem that the backup strategy is working correctly but then a failure occurs and it turns out that the existing backups do not allow the best restore operations to be performed.

For example, a backup strategy that includes a weekly full database backup plus transaction log backups every 30 minutes seems at first glance to be a sound strategy. This scenario is illustrated in Figure 1.

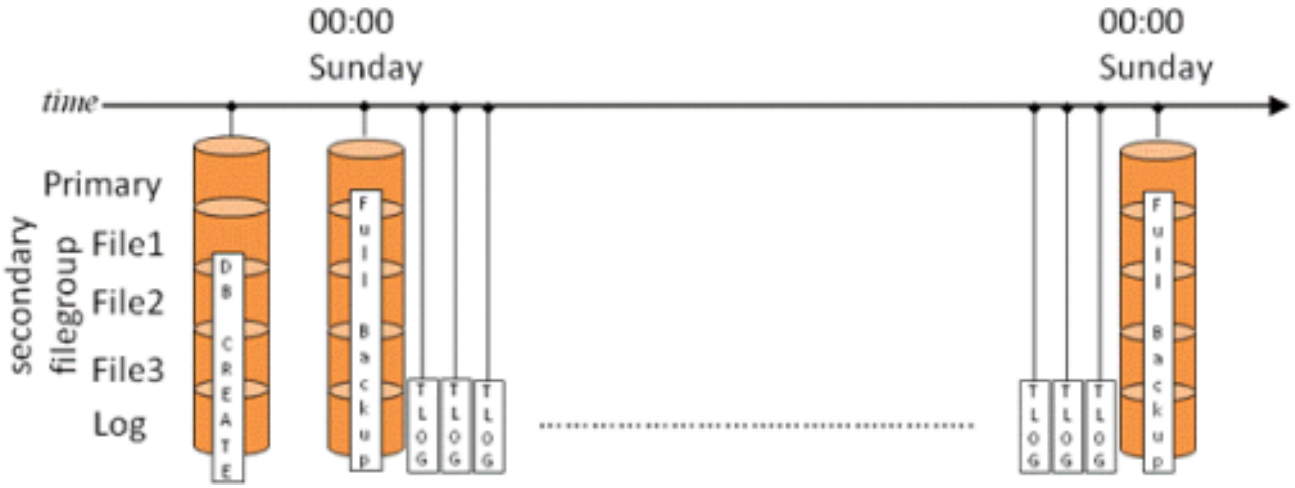

Figure 1: Backup strategy with full database and transaction log backups

However, consider the case in which a failure occurs just before the next full database backup is due to occur. The restore sequence is the previous full database backup, plus all transaction log backups since then (approximately 350!).

Although this strategy would allow recovery up to the point of the failure with no data loss, it would involve a significant amount of downtime, because all the transactions since the full database backup would effectively have to be replayed (as part of restoring the transaction log backups).

A better backup strategy, which would allow recovery with no data loss and a lot less downtime, may be to add in periodic differential database backups. A differential database backup contains all parts of the database that have changed since the previous full database backup. This is effectively the same information contained in all the transaction log backups since the previous full database backup, but is far faster to restore. This

strategy is illustrated in Figure 2.

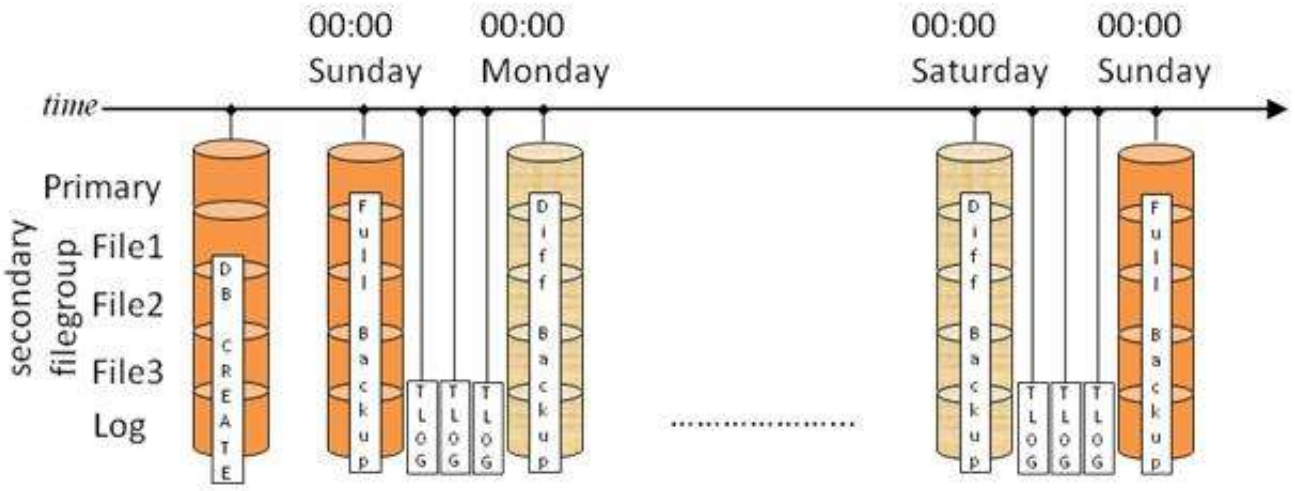

Figure 2: Backup strategy involving full database, differential database, and transaction log backups

If a failure occurs just before the next full database backup, this strategy allows the restore sequence to be the previous full database backup, the most recent differential database backup, and then all the transaction log backups since the most recent differential database backup. This is significantly fewer backups to have to restore. Of course, if the change rate of the database is very high, it may make more sense to implement more frequent full database backups instead.

The point of this example is that it is always more prudent to plan a restore strategy than it is to plan a backup strategy—work out what restore operations are required, and then let that information dictate the backup strategy. The best backup strategy is the one that best supports the restore operations the organization requires.

The details of how backup and restore operations work are beyond the scope of this white paper, but there is a comprehensive TechNet Magazine article that goes into much greater depth:

"Understanding SQL Server Backups" from the July 2009 issue (available at http://technet.microsoft.com/en-us/magazine/2009.07.sqlbackup.aspx [http://technet.microsoft.com/en-us/magazine/2009.07.sqlbackup.aspx])

SQL Server 2008 Books Online also has a large amount of information, starting with the topic "Backing Up and Restoring Databases in SQL Server" (available at http://msdn.microsoft.com/en-us/library/ms187048.aspx [http://msdn.microsoft.com/en-us/library/ms187048.aspx]).

Partial Database Availability and Online Piecemeal Restore

SQL Server 2008 includes many technologies that can dramatically reduce the time required to perform a restore operation. The most important of these technologies is partial database availability—the ability to have parts of the database offline and continue processing using the online portions.

This ability is based on having the various portions of data stored in separate filegroups. As long as the primary filegroup and the transaction log are online, the database itself is online and all other online filegroups can be used for regular processing.

For example, consider an auto parts sales database that consists of 5 sets of data—one set of tables that stores details of the various parts, and one set of tables for each US state in which the company operates. If all of the tables are stored in a single filegroup, and if that filegroup is offline, the entire sales application is also offline. If each state's tables are stored in a separate filegroup—manual partitioning of your data by region—the loss of a single filegroup may not take the entire sales application offline. Figure 3 shows this scenario.

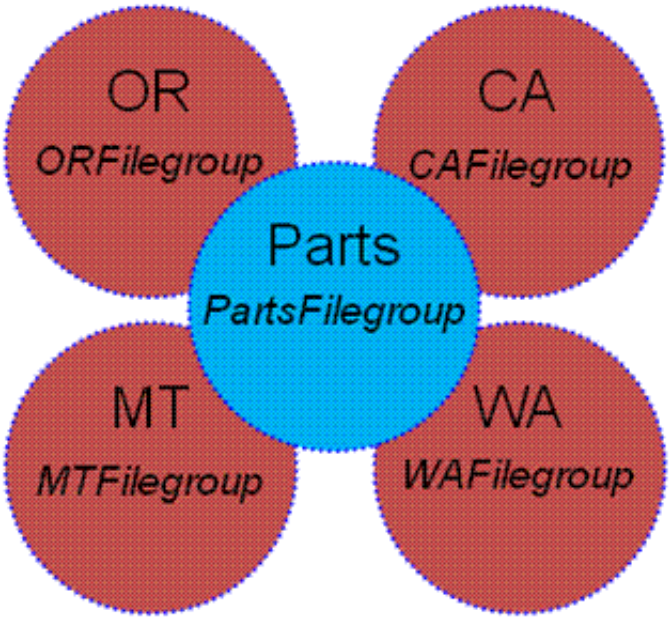

Figure 3: An example of manual partitioning

Continuing the example, if any of the state filegroups are offline, only sales to that state are affected. It is only if the Parts filegroup goes offline that the entire sales application may become unavailable, depending on the application logic. The filegroup that is damaged can be restored while the rest of the database is being used for processing, using a feature called *online piecemeal restore*.

The second way of splitting up the database is to use horizontal partitioning of tables and indexes. In this case, a single table can be split into partitions and each partition stored in a separate filegroup (or groups of partitions stored in a single filegroup – for example all months within a filegroup that contains an entire year, if the table contains data from multiple years).

As an example of horizontal partitioning, consider a sales database with a single sales table, containing all sales from the last year-to-date. If the table is stored in a single filegroup, the sales application is offline if that single filegroup is offline. However, if the table is split into multiple data ranges (for example, a calendar month in each range), with each range contained in a separate partition stored in its own filegroup, if one filegroup is offline, the sales application might not be affected at all. If the filegroup containing the current month partition is offline, no new orders can be processed, but older orders can be examined. If one or more of the filegroups containing prior month partitions is offline, new orders can be processed, and only the ability to examine older orders is compromised.

In either case, the sales table as a whole remains available for processing, with just the offline portions unavailable. As with manual partitioning, the offline portions of the sales table can be restored online while the other portions are available for processing—making use of online piecemeal restore.

This strategy also allows better control of how hardware resources are allocated. Because current sales are the most critical, these can be placed on storage arrays with better redundancy, while older sales data can be placed on storage arrays with less redundancy—saving money overall.

For more information about using partitioning, see the white paper "Partitioned Table and Index Strategies Using SQL Server 2008" (http://msdn.microsoft.com/en-us/library/dd578580.aspx [http://msdn.microsoft.com/en-us/library/dd578580.aspx]) and the SQL Server Books Online topic "Partitioning" (http://msdn.microsoft.com/en-us/library/ms178148.aspx [http://msdn.microsoft.com/en-us/library/ms178148.aspx]). For more information about partial database availability, see the white paper "Partial Database Availability" (http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/PartialDBAvailability.doc [http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/PartialDBAvailability.doc]). For more information about online piecemeal restore, see "Performing Piecemeal Restores" (http://msdn.microsoft.com/en-us/library/ms177425.aspx [http://msdn.microsoft.com/en-us/library/ms177425.aspx]) in SQL Server Books Online.

As one more example of piecemeal restore, it is also possible to restore a single database page from a backup, as long as transaction log backups exist from the time of the last full database backup up to the current time. The ability to restore only that page (or pages) that are damaged, and to do it online, further reduces the downtime required to recover from a failure.

For more information about single-page restore, see "Performing Page Restores" (http://msdn.microsoft.com/en-us/library/ms175168.aspx [http://msdn.microsoft.com/en-us/library/ms175168.aspx]) in SQL Server Books Online.

Instant File Initialization

When a restore operation is performed, the first phase of the restore is to create the database files being restored (if they do not already exist). The second phase is for SQL Server to zero-initialize the newly created database data files for security purposes. The restore operation cannot continue until the zero-initialization completes.

Zero-initialization involves sequentially writing blocks of zeroes into the data files to overwrite the previous contents of that area of the NTFS volume. The time required for this initialization is therefore directly proportional to the data file size. If the restore operation is being performed as part of disaster recovery, and the database is offline while the restore is being performed, the zero-initialization process could prevent the RTO from being reached.

For example, with a 1-terabyte data file and a backup containing 120 gigabytes (GBs) of data, the entire 1-terabyte data file must be created and zero-initialized, even though only 120 GBs of data will be restored into it.

In all editions of SQL Server 2008, the zero-initialization process can be skipped entirely using a feature called *instant file initialization*. Through the granting of a Windows® privilege to the SQL Server service account, the SQL Server instance creates the file and instructs NTFS that it does not need to be zero initialized. This change can realize a dramatic drop in the time required to perform a restore operation, if the database data files must be created as part of the restore. This can also help during database mirroring setup, when the initial full database backup is restored.

After it is enabled, this feature also allows regular creation and expansion of database data files to skip the zero-initialization process, thus providing further availability gains. It should be noted that this feature does not apply to the database transaction log file—transaction log files must always be zero initialized to guarantee correct operation of the crash recovery process.

For more information about this feature, see "Database File Initialization" (http://msdn.microsoft.com/en-us/library/ms175935.aspx [http://msdn.microsoft.com/en-us/library/ms175935.aspx]) in SQL Server Books Online.

Mirrored Backups

It is always advisable to have multiple copies of backups, in case a backup becomes damaged in some way by an I/O subsystem failure. For instance, if only a single copy of a damaged transaction log backup exists and there is no other backup spanning that time period, the database cannot be restored past the time period of the damaged transaction log backup.

SQL Server 2008 allows a backup to be mirrored to up to three local or remote locations (for a total of four copies of the backup) in the same Transact-SQL statement. This can reduce the complexity of a multiple-step copy process and can provide assurance that when a backup operation completes, the redundant copy (or copies) already exist.

For more information, see "Using Mirrored Backup Media Sets" (available at http://msdn.microsoft.com/en-us/library/ms175053.aspx [http://msdn.microsoft.com/en-us/library/ms175053.aspx]) in SQL Server Books Online.

Backup Checksums

It is prudent to ensure that a backup does not contain data that is already corrupt and also to regularly verify the integrity of existing backups. It is far better to find that a backup is corrupt or contains corrupt data *before* it is required during disaster recovery, so that a new backup can be taken.

The backup checksum feature enhances the ability of SQL Server 2008 to perform both of these operations. When backup checksums are enabled, all database data pages containing a page checksum are validated against that checksum when they are read from the database data file(s) before being written into the backup. If the page is found to be corrupt, the backup fails by default. Also, the backup checksum feature calculates and stores a separate checksum over the entire backup.

Both sets of checksums can be validated after the backup has completed using the RESTORE VERIFYONLY Transact-SQL command. This command retests all the data page checksums within the backup and

recalculates the checksum over the entire backup. If any corruptions have occurred to the backup, one of these tests will fail, giving an early indication that the backup integrity has been compromised.

Backup Compression

A comprehensive backup strategy can lead to several challenges:

- How to reduce the storage capacity required to store multiple backups for long periods of time
- How to reduce the time required for backup operations (that is, to reduce the time during which there is extra I/O load)
- How to reduce the time required for restore operations (that is, to reduce the overall downtime)

The backup compression feature of SQL Server 2008 helps to solve all these problems. It allows all types of backups to be compressed during the backup process. The tradeoff is that backup compression requires extra CPU resources, as with any compression technology. If this tradeoff poses a problem, the Resource Governor feature (discussed later) can be used to limit the amount of CPU that the backup operation consumes. For more information, see "How to: Use Resource Governor to Limit CPU Usage by Backup Compression (Transact-SQL)" (http://msdn.microsoft.com/en-us/library/cc280384.aspx [http://msdn.microsoft.com/en-us/library/cc280384.aspx]) in SQL Server Books Online.

The size of a backup is reduced by whatever compression ratio can be achieved, dependent on the data being compressed. The time required for the backup operation is reduced because the amount of data that must be written to the backup file(s) is reduced. The time required for the restore operation is reduced because the amount of data that must be read from the backup file(s) is reduced.

In terms of high availability, the most important benefit is the reduction in restore time, because this translates into a reduction in overall downtime during disaster recovery. Backup compression also forces backup checksums to be calculated, further ensuring that the backups do not contain corrupt data.

Backup compression is only available with SQL Server 2008 Enterprise, but all editions of SQL Server 2008 can restore a compressed backup.

For more information, see "Backup Compression" (http://msdn.microsoft.com/en-us/library/bb964719.aspx [http://msdn.microsoft.com/en-us/library/bb964719.aspx]) in SQL Server Books Online and the following white papers:

- "Tuning the Performance of Backup Compression in SQL Server 2008" (http://sqlcat.com:80/technicalnotes/archive/2008/04/21/tuning-the-performance-of-backup-compression-in-sql-server-2008.aspx [http://sqlcat.com/technicalnotes/archive/2008/04/21/tuning-the-performance-of-backup-compression-in-sql-server-2008.aspx])

- "Tuning Backup Compression Part 2" (http://sqlcat.com:80/technicalnotes/archive/2009/02/16/tuning-backup-compression-part-2.aspx [http://sqlcat.com/technicalnotes/archive/2009/02/16/tuning-backup-compression-part-2.aspx])

Other Single-Instance Technologies

Backups (and the related technologies discussed earlier) are certainly the most important part of a high-availability strategy if only a single instance of SQL Server 2008 can be employed. However, there are other technologies that can enhance the ability of the DBA to maintain high-availability—these will be discussed in this section.

Online Operations

A database cannot be put into production and then disregarded—if data changes, maintenance is usually required. The most common database maintenance operations include:

- Taking backups.
- Proactively checking for I/O subsystem corruption.
- Addressing index fragmentation.

SQL Server 2008 allows all these operations to be performed online, that is, without causing long-term blocking of other operations on the database.

Backup operations have always been online—there are no backup operations that cause any blocking due to

locks or require a specific maintenance window. It is true that a backup operation causes additional I/O and CPU resources to be used, and on heavily-loaded systems, this resource contention could be mistaken for the backup process blocking normal operations.

As discussed previously, restore operations may or may not require exclusive access to the database, depending on the restore being performed and the physical layout (that is, files and filegroups) of the database.

Proactive corruption detection is best done with a combination of page checksums and regular consistency checks using the Transact-SQL DBCC CHECKDB command. Page checksums are enabled by default for databases created with SQL Server 2008 and incur a negligible resource overhead to calculate and verify. The DBCC CHECKDB command (and derivative commands) has been an online operation since SQL Server 2000.

Index fragmentation (whether in a clustered index or a nonclustered index) can be removed in two ways—rebuilding an index or reorganizing an index. There are pros and cons to each method, which are beyond the scope of this white paper, but it is always possible to address index fragmentation online using one or both of these methods, depending on the table/index schema.

Index reorganizing has been an online operation since SQL Server 2000. SQL Server 2005 Enterprise Edition added the ability to create, drop, rebuild, move, and (re)partition indexes online.

For more information about online index operations, see the white paper "Online Index Operations in SQL Server 2005" (http://msdn.microsoft.com/en-us/library/cc966402.aspx [http://msdn.microsoft.com/en-us/library/cc966402.aspx]).

If the database employs partitioning (as described previously), some of these maintenance operations can be performed on a single partition (or filegroup). This can vastly reduce the resources and time required to perform the operation. For instance, continuing the sales table example described earlier, the partition holding the current month's sales may be the only one requiring fragmentation removal (as the partitions containing the prior months' data may be read-only), and so the entire table does not have to be processed as part of the operation.

However, online index rebuilds cannot be performed at the partition level so often a more complex architecture is used to better handle high availability. At the simplest level, this architecture involves two tables—one holding the current, read/write data and one holding the older, read-only data. The read/write table has only a single partition and the read-only table has multiple partitions (for example, by month).

Using this architecture, online index rebuilds can be performed on the read/write data, because the operation affects the entire table, not a single partition of a multipartition table. Because the read-only data does not change, it does not require index maintenance.

Furthermore, this architecture can be used to simplify indexing strategies. The read/write table can have only the minimum number of indexes required to support OLTP operations, and the read-only table can have as many indexes as necessary to support data analysis. This means that operations on the current data do not incur the overhead of having to maintain indexes that are not useful for OLTP operations.

Under this architecture, operations can be performed on the entire set of data using a feature called *partitioned views*—essentially performing a union operation on the two tables before querying.

While complex architectures such as combining table partitioning with partitioned views are beyond the scope of this article, they describe why database maintenance must always be considered when architecting a highly-available database. For more information about database maintenance, and the technologies described here, see the TechNet Magazine article "Top Tips for Effective Database Maintenance" (http://technet.microsoft.com/en-us/magazine/cc671165.aspx [http://technet.microsoft.com/en-us/magazine/cc671165.aspx]).

Database Snapshots

A database snapshot is a point-in-time, transactionally-consistent view of a database. After the database snapshot is created from the source database, any pages that are changed in the source database are copied synchronously into the database snapshot before the change is reflected in the source database (commonly known as *copy-on-write* but technically *copy-before-write*).

The database snapshot only contains the prechange copies of the pages from the source database that have changed since the database snapshot was created. These pages, plus all the unchanged pages in the source database, provide the point-in-time image of the source database. The database snapshot grows according to the proportion of the source database that is changed after the database snapshot is created.

A read operation performed on the database snapshot may retrieve pages from the database snapshot and the source database to satisfy the read request. Figure 4, from SQL Server 2008 Books Online, shows how read operations on the database snapshot work depending on which pages are in the database snapshot or not.

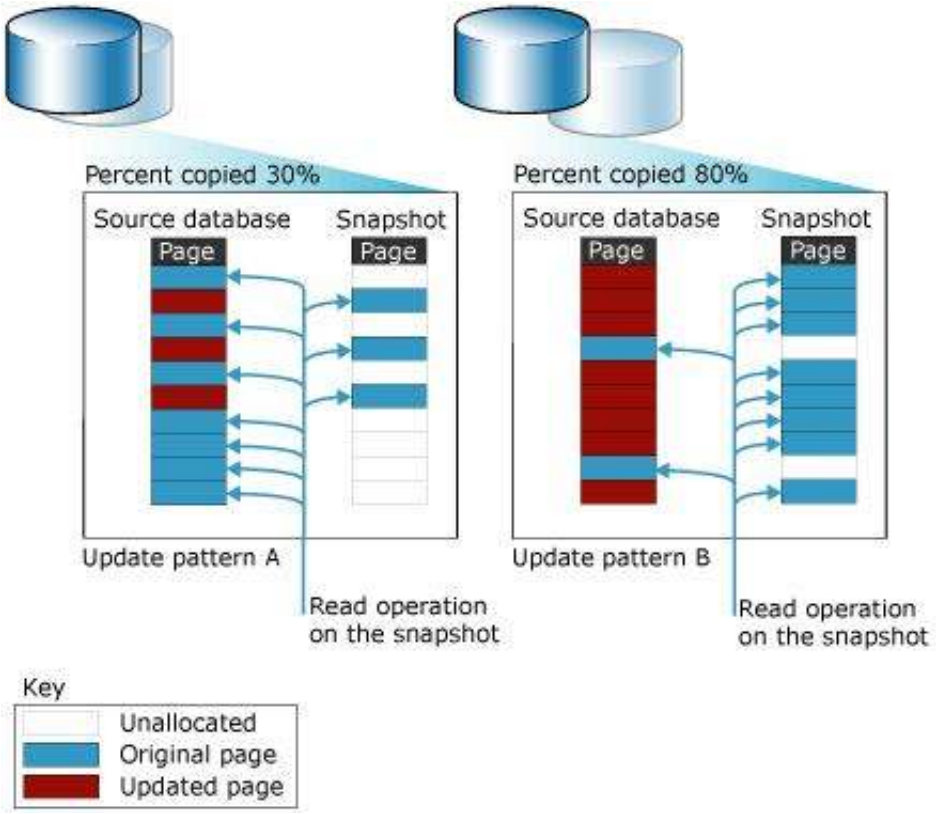

Figure 4: Example read operations on a database snapshot

Because the database snapshot does not contain the entire source database, just those pages that have changed since the database snapshot was created, it is exceptionally space-efficient. It only requires as much disk space as is needed to store the pages it contains. It is a common misconception that the database snapshot reserves the maximum amount of space needed, or checks that the maximum amount of space is available when being created—it does not.

A database snapshot has a number of uses, including:

- Providing a "point-in-time" copy of a database for end-of-business-cycle reporting.
- Allowing a mirrored database to be accessed for querying or reporting.
- Providing a safeguard against accidental data loss or other mistakes by having a copy that can be used for recovering the deleted data.

It is the last item on the list that makes a database snapshot a useful addition to a high-availability strategy. For example, consider a production database where the users have direct access to the database and are able to execute DDL commands. If a user were to accidentally drop a critical table, how could it be recovered? In most cases, the database backups must be restored to another location, and then the deleted data must be recovered from there.

However, if a database snapshot exists from before the time that the table was dropped, the table still exists in the database snapshot, because it provides an unchanging image of the database *at the time the database snapshot was created*. The deleted table can be recovered from the database snapshot, or the entire database can be reverted to the time the database snapshot was created. This can be a far faster way to recover the data and bring the application online again than restoring from backups, albeit with the potential that all changes that occurred after the snapshot was created are lost.

As a further example, consider a DBA who is about to make some complex changes to a production database. As a safeguard against making a mistake, the DBA may create a database snapshot of the production database as a way of being able to undo all the changes without having to restore the entire database from backups.

Database snapshots may increase the I/O workload for a SQL Server 2008 instance, so care must be taken when considering how to use them in an overall high-availability strategy. This applies especially when a database snapshot is used in conjunction with database mirroring, because the extra I/O workload may cause the database mirror to lag behind the principal, leading to additional downtime required during a failover operation.

It should also be noted that database snapshots are not a substitute for a comprehensive backup strategy. If the source database becomes corrupt or goes offline, the database snapshot is similarly affected (the converse is never true, however). Because of this linkage, a database snapshot cannot be used to protect against loss of the entire database, nor can it be used to perform more targeted disaster recovery operations such as page restore.

For more information, see "Database Snapshots" (http://msdn.microsoft.com/en-us/library/ms175158.aspx [http://msdn.microsoft.com/en-us/library/ms175158.aspx]) in SQL Server Books Online.

Hot-Add Memory and CPU

In SQL Server 2008, you can add more memory and CPUs to the server hosting the SQL Server instance (whether physically, logically, or virtually). You can then reconfigure SQL Server 2008 online to make use of the new resources available. This enables the database workload to continue completely uninterrupted while the hardware upgrade proceeds.

For more information about these technologies and the Windows and CPU architecture requirements, see the following SQL Server 2008 Books Online topics:

- "Hot Add Memory" (http://msdn.microsoft.com/en-us/library/ms175490.aspx [http://msdn.microsoft.com/en-us/library/ms175490.aspx])

- "Hot Add CPU" (http://msdn.microsoft.com/en-us/library/bb964703.aspx [http://msdn.microsoft.com/en-us/library/bb964703.aspx])

Resource Governor

SQL Server 2008 introduces a new technology, Resource Governor, which provides you with more control over query execution memory and CPU usage, at a variety of granularities (for example, an individual connection to SQL Server, or several groups of users in aggregate). The aim of the technology is to enable scenarios such as:

- Guaranteeing a certain level of resources for critical workloads, thus maintaining performance within prescribed limits.
- Limiting the amount of resources that ad-hoc queries can use, thus preventing an ad-hoc query from using all resources on a SQL Server 2008 instance and essentially taking the instance offline for the duration of the query.
- Limiting the amount of resources that maintenance operations can use, thus preventing a long-running maintenance operation from degrading regular workload performance (for instance, as described earlier with backup compression).

In a high-availability strategy, Resource Governor is a different kind of preventative technology, in that it does not provide any protection against the usual causes of downtime. Instead it enables the DBA to configure SQL Server to protect against unforeseen resource contention that may manifest itself to users as downtime.

For more information, see "Managing SQL Server Workloads with Resource Governor" (http://msdn.microsoft.com/en-us/library/bb933866.aspx [http://msdn.microsoft.com/en-us/library/bb933866.aspx]) and the white paper "Using the Resource Governor" (http://download.microsoft.com/download/D/B/D/DBDE7972-1EB9-470A-BA18-58849DB3EB3B/ResourceGov.docx [http://download.microsoft.com/download/D/B/D/DBDE7972-1EB9-470A-BA18-58849DB3EB3B/ResourceGov.docx]).

Multi-Instance Technologies

Most high-availability strategies include a technology to maintain a redundant copy of the database and/or redundant hardware on which to run the SQL Server instance. SQL Server 2008 provides four technologies to do this, with varying characteristics and abilities.

Log Shipping

Log shipping is the simplest way to provide one or more redundant copies of a single database. The primary database on the primary server is backed up and then restored to one or more secondary servers. Transaction log backups are then repeatedly taken of the primary database, shipped (that is, copied over the network) to the secondary server(s), and then restored. In this manner, the secondary databases are continually being updated with changes from the primary database through transaction log restores. An optional third SQL Server instance can be used to monitor the primary and secondary servers to ensure that transaction log backups are being taken and restored regularly.

There are various configuration options, such as how often a transaction log backup is taken on the primary server and how often the transaction log backups are restored on the secondary server(s). A log shipping secondary server can also be set to have a load delay (that is, a period of time to wait before restoring a transaction log backup). This is very useful for keeping an older copy of the database to allow for recovery from human error. For instance, if a log shipping secondary server is set to have an 8-hour load delay and someone accidentally deletes a table from the primary database within this period, the table still exists in the secondary server and can be recovered.

A log shipping secondary can also be set to allow read-only access to the database in between transaction log restore operations. This enables the database to be used for reporting purposes, maximizing use of the redundant server, albeit with some extra configuration.

Log shipping can be thought of simply as backup, copy, restore, repeat. Because log shipping involves a multistage process that cannot guarantee zero data loss, and it does not have any built-in mechanism to automate failover from the primary server to the secondary server, it is termed a *warm standby* solution.

The downtime involved in bringing a log shipping secondary server online varies depending on whether there are more transaction log backups to restore and whether the database must be brought to as recent a point as possible. Furthermore, neither the detection of failure of the primary server nor the failover to a secondary server are automatically detected by client applications, so extra logic must be added to the client or possibly in a mid tier.

The log shipping process is controlled through a series of SQL Server Agent jobs that perform the backups, copies, restores, and monitoring. Figure 5 shows a log shipping configuration with the primary server instance, three secondary server instances, and a monitor server instance.

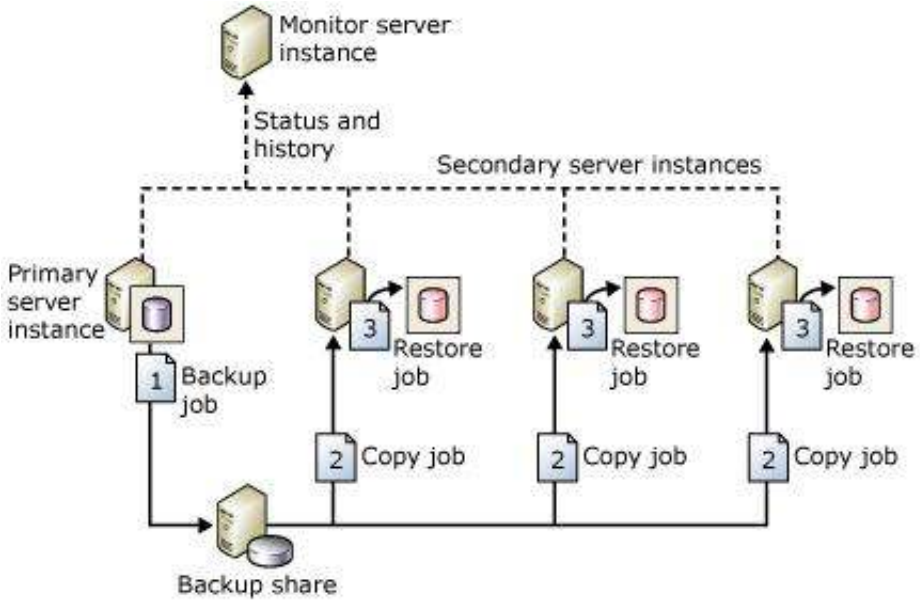

Figure 5: Example log shipping configuration

The figure illustrates the steps performed by backup, copy, and restore jobs, as follows:

- The primary server instance runs the transaction log backup job on the primary database. This instance then places the log backup into a primary log-backup file, which it sends to the backup folder. In this figure, the backup folder is on a shared directory, known as the .
- Each of the three secondary server instances runs its own copy job to copy the primary log-backup file to its own local destination folder.

- Each secondary server instance runs its own restore job to restore the log backup from the local destination folder onto the local secondary database.

The primary and secondary server instances send their own history and status to the monitor server instance.

For more information about log shipping, see "Log Shipping" (http://msdn.microsoft.com/en-us/library/bb895393.aspx [http://msdn.microsoft.com/en-us/library/bb895393.aspx]).

Transactional Replication

Replication is a broad set of technologies that enable data to be copied and distributed between servers and then synchronized to maintain consistency. The replication technologies applicable to a high-availability strategy are *transactional replication* and its derivative, *peer-to-peer replication*.

Transactional replication involves creating a publication (data in one or more tables in a database) in a publication database on a Publisher instance. The initial state of the publication is copied to one or more Subscriber instances where it becomes the subscription in a subscription database. This initialization process can be performed using a database backup or using a snapshot (in which replication itself works out what needs to be copied and only copies what is needed, rather than creating a full database backup).

After the subscriptions are initialized, transactions are replicated from the Publisher to the Subscribers, using the process shown in Figure 6.

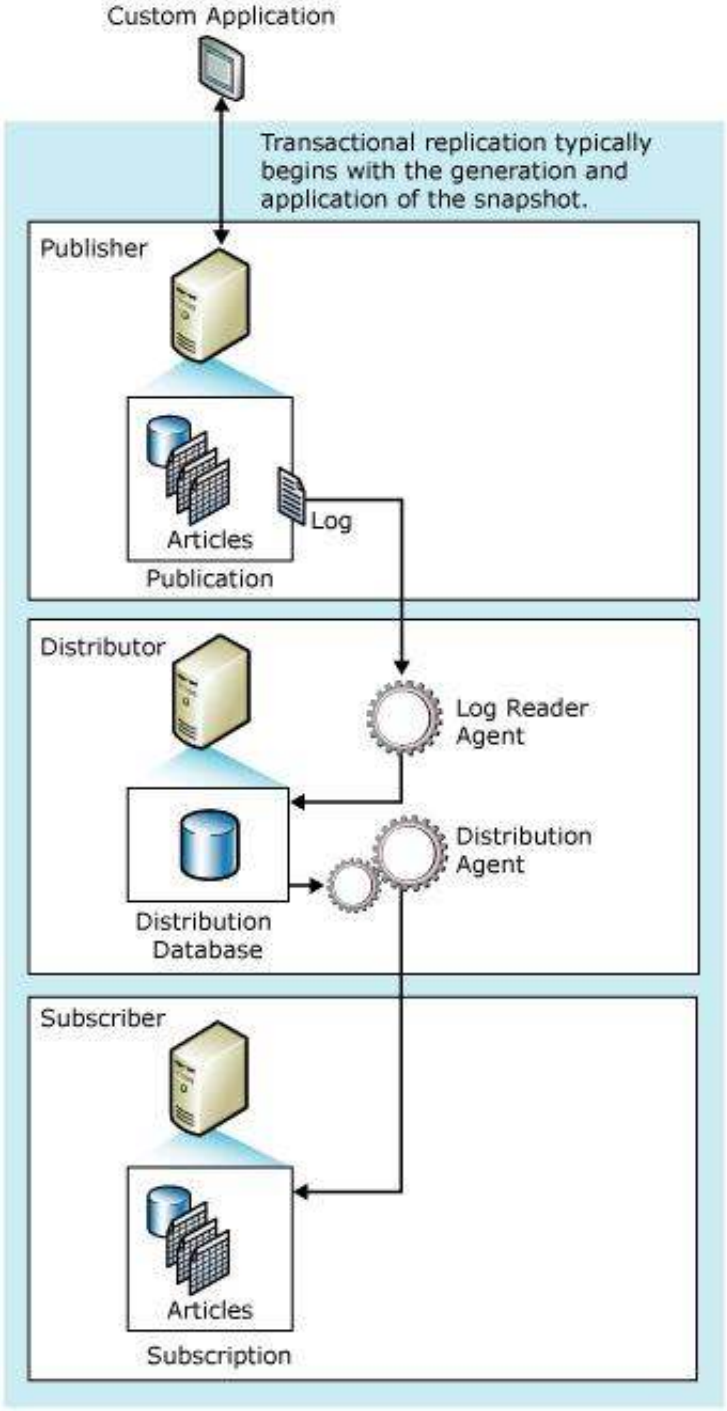

Figure 6: Transactional replication process

Committed transactions that affect the publication are read from the transaction log on the Publisher, stored in the distribution database on the Distributor, and then applied to the subscription database on the Subscriber(s). In this way the subscriptions are constantly being updated with changes that occur to the publication. If the Publisher goes offline, transactional replication automatically restarts when it comes online again. Also, if a Subscriber goes offline, the transactions that must be propagated to it are held in the distribution database until the Subscriber comes online again.

The intermediate storage of the transactions is necessary so that the transactions do not have to be stored in the transaction log of the publication database. Ideally the Distributor is a separate SQL Server instance running on a separate server, because this provides extra redundancy and offloads the distribution workload from either the Publisher or Subscriber instances.

As with log shipping, replication does not provide automatic detection of a Publisher failure or automatic failover to a Subscriber, so it is also a warm standby solution. Additionally, there is latency between a transaction occurring on the Publisher and being propagated to the Subscribers, so data loss is possible at the time of a failure. Finally, transactional replication only protects the data in the publication—it cannot be used to

protect an entire database or group of databases.

Peer-to-peer transactional replication allows multiple servers (known as *nodes)* to act as Publishers and Subscribers for the same data. A change made on one node in the peer-to-peer topology is propagated to all other nodes, with all nodes acting as Publisher, Subscriber, and (usually) Distributor. In peer-to-peer replication, the data is highly available, and solutions are easily scaled out for query workloads.

An example peer-to-peer transactional replication topology is shown in Figure 7.

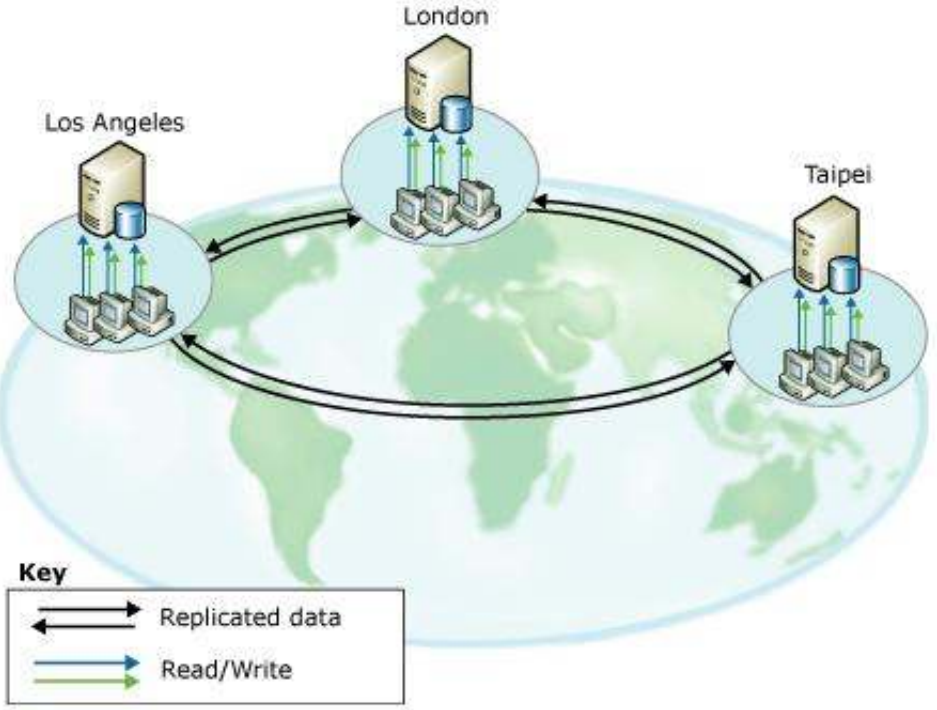

Figure 7: Example peer-to-peer transactional replication topology

The preceding illustration shows three participating databases that provide data for a worldwide software support organization, with offices in Los Angeles, London, and Taipei. The support engineers at each office take customer calls and enter and update information about each customer call. The time zones for the three offices are eight hours apart, so there is no overlap in the workday. As the Taipei office closes, the London office is opening for the day. If a call is still in progress as one office is closing, the call is transferred to a representative at the next office to open.

Each location has a database and an application server, which are used by the support engineers as they enter and update information about customer calls. The topology is partitioned by time. Therefore, updates occur only at the node that is currently open for business, and then the updates flow to the other participating databases. This topology provides the following benefits:

- Independence without isolation: Each office can insert, update, or delete data independently but can also share the data because it is replicated to all other participating databases.
- Higher availability in case of failure or to allow maintenance at one or more of the participating databases.

Peer-to-peer transactional replication does not have automatic detection or automatic failover, so the various nodes in the topology provide warm standby copies of the published data. Additionally, peer-to-peer transactional replication has the same latency issues as regular transactional replication.

As with log shipping, a failover is not be detected by client applications. This means extra logic must be added to the client or possibly in a mid tier to handle their redirection. This logic must be added separately.

For more information, see "Transactional Replication Overview" (http://msdn.microsoft.com/en-us/library/ms151176.aspx [http://msdn.microsoft.com/en-us/library/ms151176.aspx]) and "Peer-to-Peer Transactional Replication" (http://msdn.microsoft.com/en-us/library/ms151196.aspx [http://msdn.microsoft.com/en-us/library/ms151196.aspx]).

Database Mirroring

Database mirroring provides a redundant copy of a single database that is automatically updated with changes. Database mirroring works by sending transaction log records from the main database (called the *principal*) to the redundant database (called the *mirror*). The transaction log records are then replayed on the mirror database continuously (that is, the mirror database is constantly running recovery using the transaction log records sent from the principal).

The principal and mirror databases are usually hosted on separate SQL Server 2008 instances (called the *principal server* and *mirror server*) on separate physical servers, and often in separate data centers. This basic configuration is shown in Figure 8.

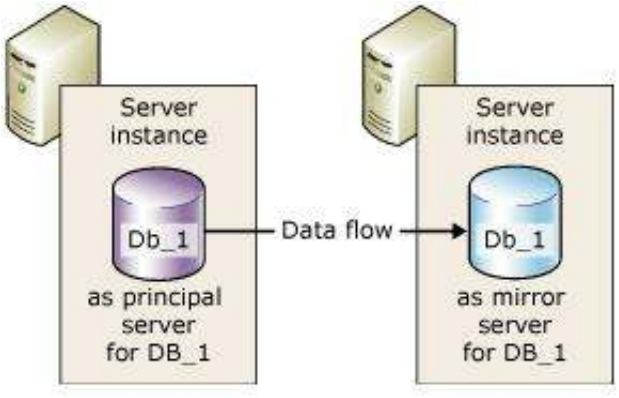

Figure 8: Basic database mirroring configuration

Application connections are only accepted to the principal database—connections attempted to the mirror database may be redirected to the principal database using client redirection, which is explained later in this section. A mirror database can be used for querying, however, through the use of a database snapshot. Additionally, database mirroring supports only a single mirror.

Database mirroring is different from log shipping or transactional replication in the following ways:

- Failures are automatically detected.
- Failovers can be made automatic.

If it is configured to support automatic failure detection and failover, database mirroring provides a hot standby solution.

There are two modes of database mirroring—synchronous and asynchronous. With synchronous mirroring, transactions cannot commit on the principal until all transaction log records have been successfully copied to the mirror (but not necessarily replayed yet). This guarantees that if a failure occurs on the principal and the principal and mirror are synchronized, committed transactions are present in the mirror when it comes online —in other words, it is possible to achieve zero data loss.

Synchronous mirroring can be configured to provide automatic failover, through the use of a third SQL Server 2008 instance called the *witness server* (usually hosted on another physically separate server). The sole purpose of the witness is to agree (or not) with the mirror that the principal cannot be contacted. If the witness and mirror agree, the mirror can initiate failover automatically. If synchronous mirroring is configured with a witness, the operating mode is known as *high-availability mode* and provides a hot standby solution. When no witness is defined, the operating mode is known as *high-safety* mode, which provides a warm standby solution.

The high-availability configuration is shown in Figure 9.

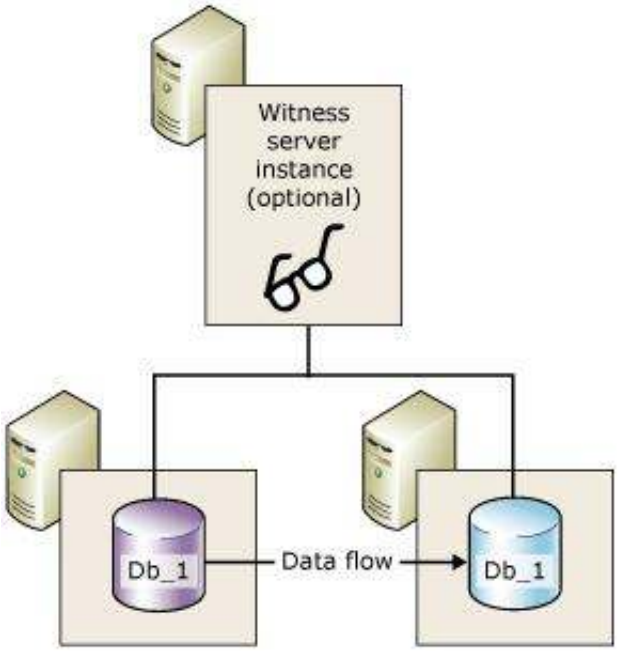

Figure 9: High-availability configuration using a witness server

With asynchronous mirroring there is no such guarantee, because transactions can commit on the principal without having to wait for database mirroring to copy all the transaction's log records. This configuration can offer higher performance because transactions do not have to wait, and it is often used when the principal and mirror servers are separated by large distances (that is, implying a large network latency and possibly lower network bandwidth). Consequently, this operating mode is also known as *high-performance mode* and provides a warm standby solution.

If a failure occurs on the principal, a mirroring failover occurs, either manually (in the high-performance and high-safety modes) or automatically (only in the high-availability mode). The mirror database is brought online after all the transaction log records have been replayed (that is, after recovery has completed). The mirror becomes the new principal and applications can reconnect to it. The amount of downtime required depends on how long it takes for the failure to be detected and how much transaction log needs to be replayed before the mirror database can be brought online.

Any transactions that are in-flight on the principal when a failure occurs are lost, as are any transactions that have committed on the principal but for which all transaction log records have not yet been sent to the mirror. The amount of data loss depends on the operating mode configured—the configuration that comes closest to guaranteeing zero data-loss is high-safety or high-availability mode where the mirror database is synchronized with the principal (that is, completely up-to-date). If another configuration is chosen, the amount of data loss depends on how much transaction log has not yet been copied from the principal to the mirror.

After a failover, the application must reconnect to the database, but the application does not know whether the (old) principal server or the mirror server is now the principal. There are two ways for the application to connect, using implicit or explicit client redirection. With explicit client redirection, the application specifies both mirroring partners in the connection string, so no matter which is the principal, the connection should succeed. With implicit client redirection, only one partner is specified, but if it is the current mirror, it redirects the connection to the current principal. However, if the specified partner is not available to perform the redirection (for example, it is completely offline), the connection fails. For this reason, we recommend always using explicit client redirection.

SQL Server 2008 adds two important features to database mirroring: automatic page repair and log stream compression.

Automatic page repair makes use of the fact that, when synchronized, the mirror database is an exact physical copy of the principal database. If a corrupt page is found in the principal database (for instance, a page checksum failure), the database mirroring system attempts to obtain the page image from the mirror database and then repair the page corruption. The mechanism also works in the other direction if a corrupt page is found on the mirror database while transaction log records are being replayed.

This mechanism can remove the need for immediate downtime as soon as a corruption is discovered, giving the system administrator time to take corrective action. It should be noted that this feature it not a substitute for recovering from corruption, just a temporary mitigation.

Log stream compression enables more transaction log to be sent through the same network link by compressing the log records before transmission to the mirror. This can reduce the performance penalty of using synchronous database mirroring and also reduce the amount of unsent transaction log when using asynchronous database mirroring (thus reducing data loss in the event of a failure).

Log stream compression requires extra CPU resources on the principal and mirror servers to perform the compression and decompression. There may also be more CPU required on the principal because a higher transaction workload can be accommodated. This feature is enabled by default, but it can be disabled if CPU resources are not available or if the compression ratio does not make the trade-off worthwhile.

For more information, see "Database Mirroring" (http://msdn.microsoft.com/en-us/library/bb934127.aspx [http://msdn.microsoft.com/en-us/library/bb934127.aspx]) and in the following white papers:

- "Database Mirroring in SQL Server 2005" (http://www.microsoft.com/technet/prodtechnol/sql/2005/dbmirror.mspx [http://www.microsoft.com/technet/prodtechnol/sql/2005/dbmirror.mspx])

- "Database Mirroring Best Practices and Performance Considerations" (http://www.microsoft.com/technet/prodtechnol/sql/2005/technologies/dbm_best_pract.mspx [http://www.microsoft.com/technet/prodtechnol/sql/2005/technologies/dbm_best_pract.mspx])

- "Alerting on Database Mirroring Events" (http://www.microsoft.com/technet/prodtechnol/sql/2005/mirroringevents.mspx [http://www.microsoft.com/technet/prodtechnol/sql/2005/mirroringevents.mspx])

Failover Clustering

Failover clustering is the only technology that allows an entire SQL Server 2008 instance to be made highly available – either at the local data center or possibly at a remote site or data center. A SQL Server failover cluster is implemented on a Windows Server failover cluster.

A failover cluster usually consists of two or more cluster nodes (up to 16 nodes are possible with Windows Server® 2008 and SQL Server 2008 Enterprise), running one or more instances of SQL Server 2008. A failover cluster can be a *single-instance cluster* with a single SQL Server 2008 instance running on one of the cluster nodes, or a *multi-instance cluster* with multiple SQL Server 2008 instances running on multiple cluster nodes. (These used to be called *active-passive* and *active-active,* respectively.)

The failover cluster presents a virtual name and virtual IP to the network—the application does not know which cluster node is hosting the SQL Server 2008 instance to which it is connecting. Multiple SQL Server 2008 instances can be hosted by the failover cluster—a single default instance and multiple named instances. After an instance fails over to another cluster node and the application reconnects, the connection is transparently routed to the new cluster node that hosts the instance.

With a single-instance cluster, the SQL Server 2008 instance runs on one of the cluster nodes. If the node crashes, the failure is automatically detected and the SQL Server 2008 instance is automatically restarted on the remaining cluster node. With a multi-instance cluster, one or more instances run on each cluster node (potentially with a hot-spare node with no instances running). When a node crashes, the failure is automatically detected and its hosted instances are automatically restarted on other nodes (in a configurable order) in the cluster.

In both cases, the amount of time before the databases are available depends on how long crash recovery takes to complete, as described previously. When a failure occurs, any active transactions are rolled back, but committed transactions are guaranteed to exist as long as no damaged occurred to the database files themselves.

Although failover clustering provides excellent server redundancy, it does not protect against data loss if the physical disks are damaged, because there is a single, shared copy of each of an instance's databases. Additional technologies must be used to ensure redundancy of both the data and the server, as discussed in the next section.

For more information, see the white paper "SQL Server 2008 Failover Clustering" (http://download.microsoft.com/download/6/9/D/69D1FEA7-5B42-437A-B3BA-A4AD13E34EF6/SQLServer2008FailoverCluster.docx [http://download.microsoft.com/download/6/9/D/69D1FEA7-5B42-437A-B3BA-A4AD13E34EF6/SQLServer2008FailoverCluster.docx]) and "Getting Started with SQL Server 2008 Failover Clustering" (available at http://msdn.microsoft.com/en-us/library/ms189134.aspx [http://msdn.microsoft.com/en-us/library/ms189134.aspx]) in SQL Server Books Online.

Combining Multi-Instance Technologies

Often a high-availability strategy calls for multiple multi-instance strategies to be combined, for added redundancy or to mitigate a shortfall of one technology.

The prime example of this is with failover clustering, which does not provide a redundant copy of the data, as explained previously. In this case, simply using database backups or log shipping may satisfy the downtime and data loss requirements. If no data loss can be tolerated, either synchronous database mirroring or synchronous SAN replication can be used. Failover clustering can also be combined with SAN replication (and a network connecting the two clusters); the resulting combination is commonly known as *geo-clustering*.

Another example is when database mirroring is used to provide a hot standby copy of a database. It may also be a requirement to protect against human error (for example, dropping a table). In this case, one of the following options can be used: creating a database snapshot periodically on the database mirror may be a viable solution, or adding a log shipping secondary copy of the same database and configuring a load delay, as explained previously.

Even when transactional replication is used as the primary high-availability technology, it may be desirable to further protect each of the servers in the replication topology. With SQL Server 2008, the publication and subscription database can be protected using database mirroring, and the distribution database must be placed on a failover cluster for protection.

Combining multi-instance technologies adds complexity to the overall system. SQL Server 2008 Books Online contains a comprehensive section that discusses the various combinations and explains some of the issues involved, along with solutions. For more information, see "High Availability: Interoperability and Coexistence" (http://msdn.microsoft.com/en-us/library/bb500117.aspx [http://msdn.microsoft.com/en-us/library/bb500117.aspx]).

There are also two white papers that delve into specific scenarios:

- SQL Server Replication: Providing High-Availability Using Database Mirroring" (http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/ReplicationAndDBM.docx [http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/ReplicationAndDBM.docx])

- "Database Mirroring and Log Shipping Working Together" (http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/DBMandLogShipping.docx [http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/DBMandLogShipping.docx])

Virtualization

Although virtualization is not a SQL Server 2008 technology, it can be used effectively to minimize downtime. In this case, the SQL Server 2008 instance is installed in a Windows Server 2008 Hyper-V™ virtual machine that runs on a physical server and accesses data stored on a SAN, along with the virtual machine image itself. Any physical server with access to the storage can run the virtual machine.

Using Windows Server 2008 R2 and Microsoft Hyper-V Server 2008 R2, running virtual machines can be moved between Windows cluster nodes transparently using the Live Migration feature, for the purposes of upgrade or server maintenance (that is, without shutting down the SQL Server 2008 instance). Also, if a virtual machine or physical node crashes, the virtual machine automatically restarts on another node.

For more information, see the white paper "Running SQL Server 2008 in a Hyper-V Environment" (http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/SQL2008inHyperV2008.docx [http://download.microsoft.com/download/d/9/4/d948f981-926e-40fa-a026-5bfcf076d9b9/SQL2008inHyperV2008.docx]).

Mitigating the Causes of Downtime and Data Loss

This section of the white paper will revisit the causes of downtime and data loss and explain how they can be mitigated using the various technologies described previously.

It should be noted that in all cases it is recommended that a comprehensive backup strategy is be employed alongside any multi-instance technologies, to protect against the chance of multiple failures destroying all copies of the database.

Database Maintenance

Downtime and performance degradation from maintenance can be mitigated as follows:

- Performing fragmentation analysis and consistency checking on a redundant database
- Using online operations to prevent long-term blocking locks
- Using backup compression to shorten the duration of backup operations
- Limiting the CPU resources of maintenance operations using Resource Governor

Batch Operations

Downtime from batch operations can be mitigated using table and index partitioning. Partitioning allows bulk-delete and bulk-insert operations to be performed as metadata-only operations (known as *partition switching*). The details of this mechanism are beyond the scope of this white paper—for more information about using partitioning, see the white paper "Partitioned Table and Index Strategies Using SQL Server 2008" (http://msdn.microsoft.com/en-us/library/dd578580.aspx [http://msdn.microsoft.com/en-us/library/dd578580.aspx]) and the SQL Server Books Online topic "Partitioning" (http://msdn.microsoft.com/en-us/library/ms178148.aspx [http://msdn.microsoft.com/en-us/library/ms178148.aspx]).

Upgrade

Downtime from upgrades can be mitigated by failover over to another SQL Server 2008 instance with a redundant copy of the data (using log shipping, database mirroring, transactional replication, or failover clustering with SAN replication), or by failing over a SQL Server 2008 instance to another node in a failover cluster. If a zero data-loss failover is required, synchronous database mirroring or failover clustering with synchronous SAN replication are usually needed.

For physical server resource upgrades, hot-add CPU and hot-add memory can be used to enable resources to be added to a server without incurring SQL Server downtime.

Data Center Loss

The technology to use to mitigate the loss of a data center depends on how many SQL Server 2008 instances and databases must be protected, and whether another data center is available.

For a single database, log shipping or database mirroring to a remote data center is adequate, depending on the data loss requirements. For multiple databases or instances, failover clustering with SAN replication (geo-clustering) is required. To add further redundancy, a third data center with log shipping of all databases can provide warm standby copies of the databases.

An alternative to these technologies is to use transactional or peer-to-peer replication if the application ecosystem is suitable for replication rather than one of the whole-database or whole-instance technologies.

If no remote data center is available, the only defense is to have a comprehensive backup strategy and store a copy of the database backups off-site so a failure does not destroy both the data center and all copies of the database backups. This solution, of course, incurs the highest amount of downtime and potential data loss.

Server Failure

Mitigation of server failure is the same as for data center loss—the choice of technology depends on how many databases and instances are involved, plus data loss requirements.

I/O Subsystem Failure

Mitigating I/O subsystem failure means providing redundancy at the data level. The first line of defense is to configure the I/O subsystem itself to incorporate redundancy using RAID. At the SQL Server level, a redundant copy of the database must usually be maintained, with the choice of technology dependent on downtime and data loss requirements.

Alternatively, if downtime requirements allow, mitigation can be provided solely through a comprehensive backup and database design strategy, configured to allow online piecemeal restores of the smallest amount of data possible. However, if the I/O subsystem has failed completely, it may not be possible to recover past the most recent transaction log backup.

As explained previously, the only technologies that can achieve zero data loss are synchronous database mirroring and synchronous SAN replication.

Human Error

Mitigating human error that affects data relies on having a copy of the database that has not been affected by the human error. The technologies that can help here are:

- Configuring a log shipping secondary with a load delay.
- Configuring a database mirroring mirror with a database snapshot.
- Configuring mirrored backups to mitigate accidental deletion.
- Configuring a backup strategy that allows targeted restores and point-in-time recovery.

Although not discussed in this white paper, the best defense against human error is prevention through properly configured security (for example, user-schema separation, use of DDL triggers, or data manipulation through controlled stored procedures only).

High-Availability Features Supported by SQL Server 2008 Editions

Table 1 shows which SQL Server 2008 editions support the features discussed in this white paper, in the order they were discussed.

Feature name	Enterprise	Standard	All other editions
Fast recovery	Yes		
Partial database availability	Yes		
Online piecemeal restore	Yes		
Partitioning	Yes		
Instant file initialization	Yes	Yes	Yes
Mirrored backups	Yes		
Backup checksums	Yes	Yes	Yes
Backup compression	Yes		
Online index operations	Yes		
Database snapshots	Yes		
Hot-add memory	Yes		
Hot-add CPU	Yes		
Resource Governor	Yes		
Log shipping	Yes	Yes	Yes, except Express
Transactional replication	Yes	Yes	Subscriber only
Peer-to-peer transactional replication	Yes		

Database mirroring	Yes	Yes, asynchronous only	Witness only
Automatic page repair from the mirror	Yes	Yes	
Failover clustering	Operating system maximum supported nodes	2 nodes	

Table 1: High-availability features supported by the editions of SQL Server 2008

Conclusion

This white paper has described the high-availability technologies of SQL Server 2008 and how they can be used as part of a high-availability strategy, as well as some guidance on how to begin designing a high-availability strategy. To conclude, it is worth reiterating some of the major points made during the white paper.

Designing a successful high-availability strategy means understanding all the requirements the strategy needs to meet. Equally important is an understanding of the limitations that will be imposed on the implementation of the strategy. A compromise must be reached before moving forward with technology evaluation, and a post-compromise, ordered list of requirements must be used when choosing which technologies to use.

A high-availability strategy should never solely rely on multi-instance technologies—a comprehensive backup strategy should always be employed to protect against total data loss. Furthermore, backup checksums should be used and periodic integrity checking of databases and backups should be performed to ensure that backups are valid and can be used when needed. Instant initialization and backup compression should be enabled to speed up backups and restores, and larger databases should be designed to allow the smallest possible online piecemeal restores to be performed.

All four multi-instance technologies provide redundancy, but at different levels (data, database, or instance), and with different levels of transactional currency, downtime, and data loss. These differences should be carefully considered during evaluations of the technologies to ensure that the technologies chosen meet requirements of the strategy. Although it is common to combine technologies, care must be taken when doing so to avoid unforeseen side-effects.

Finally, SQL Server 2008 provides all the technologies needed to implement a high-availability strategy, but knowing you have implemented a successful high-availability strategy relies on testing the system against failures—it is far better to simulate a failure with all staff on hand to aid with recovery than to experience a failure when no one expects it and end up with more downtime and data loss than necessary.

For more information:

http://www.microsoft.com/sqlserver/ [http://www.microsoft.com/sqlserver/default.aspx] : SQL Server Web site

http://technet.microsoft.com/en-us/sqlserver/ [http://technet.microsoft.com/en-us/sqlserver/default.aspx] : SQL Server TechCenter

http://msdn.microsoft.com/en-us/sqlserver/ [http://msdn.microsoft.com/en-us/sqlserver/default.aspx] : SQL Server DevCenter

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

- Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
- Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback [mailto://microsoft.com:25/default.aspx?subject=White%20Paper%20Feedback:%20High%20Availability%20with%20SQL%20Server%202008] .