Microsoft® SQL Server™ 2005

Database Mirroring Best Practices and Performance Considerations

SQL Server Technical Article

Writer: Sanjay Mishra, Microsoft Corp.

Technical Reviewers: Brian Goldstein, Microsoft Corp.
Lubor Kollar, Microsoft Corp.
Mike Ruthruff, Microsoft Corp.
Steven Wort, Microsoft Corp.
Prem Mehra, Microsoft Corp.
Peter Byrne, Microsoft Corp.
Mark Wistrom, Microsoft Corp.
Jakub Kulesza, Microsoft Corp.
Juergen Thomas, Microsoft Corp.
Michael Raheem, Microsoft Corp.
Allan Hirt, Avanade, Inc.

Published: February 2006

Applies To: SQL Server 2005

Summary: Maximizing the availability of databases is on the top priority list for many database administrators worldwide. Database mirroring is a new feature in SQL Server 2005 that can help minimize planned and unplanned downtime, thereby maximizing the availability of the database. Database mirroring transports the changes in the production database to a mirror database, either synchronously or asynchronously. The mirror database can reside either in the same data center to provide a high-availability solution, or in a remote data center to provide a disaster-recovery solution. Business requirements such as service-level agreements and performance, as well as technical factors such as log generation rate, network throughput, and I/O throughput, influence the deployment of database mirroring. This paper discusses best practices and performance considerations for implementing database mirroring.

Copyright

Table of Contents

Introduction

One of the most important requirements of business-critical applications is the availability of the database. Maximizing the availability of the database is on the top priority list of many database administrators. Database mirroring, a new feature in Microsoft SQL Server™ 2005, adds another alternative to the SQL Server availability arsenal.

To maximize database availability, unplanned as well as planned downtime needs to be minimized. Unplanned downtime is primarily caused by hardware failure (computer failure and storage failure), disk corruption, power outages, communication failures, natural disasters, terrorism, human error, and other factors that cause the primary production database, the production server, and/or the production data center to be unavailable. Planned downtime is primarily due to changes that are applied to the production system. These might be hardware upgrades, software upgrades, and database storage and configuration changes, which cause the primary database or server to be unavailable for a short period of time.

Database mirroring helps minimize both planned and unplanned downtime by:

- Maintaining a mirror database that is kept up-to-date with the production database either synchronously or asynchronously.
- Providing ways to perform automatic as well as manual failover.
- Allowing the mirror database to be in a remote data center, thereby providing a foundation for disaster recovery.

This white paper presents database mirroring best practices and performance considerations. The target audiences for this white paper are:

- Database administrators who are implementing database mirroring.
- Database analysts and system administrators who are designing a highly available database application or a framework for disaster recovery.

Prior to reading this white paper, readers should consider the following reading as prerequisites:

- "Database Mirroring" in SQL Server 2005 Books Online.
- The Database Mirroring in SQL Server 2005 white paper on Microsoft TechNet.
- The Microsoft support note Issues to consider when you use the database mirroring feature in SQL Server 2005.

Database Mirroring Concepts

Database mirroring maintains a hot standby database (known as the mirror database) that can quickly assume client connections in the event of a principal database outage. Database mirroring involves two copies of a single database that reside on different computers. At any given time, only one copy of the database is available to clients. This copy is known as the principal database. Unlike log shipping which works by applying full transaction log backups to a warm standby database, database mirroring works by transferring and applying a stream of database log records from the principal database to the other copy of the database (the mirror database). Database mirroring applies every database modification that is made on the principal database to the mirror database. This includes data changes as well as changes to the physical and logical structures in the database, such as database files, tables, and indexes.

For a detailed discussion of database mirroring concepts, see SQL Server 2005 Books Online. Following are some basic database mirroring terms.

Principal: In a database mirroring configuration, there are two copies of a single database, but only one copy is accessible to the clients at any given time. The copy of the database that the applications connect to is called the *principal database*. The server that hosts the principal database is known as the *principal server*.

Mirror: The *mirror* is the copy of the principal database. The mirror is always in a restoring state; it is not accessible to the applications. To keep this database up-to-date, the log records are transferred from the principal and applied on the mirror database. The server that hosts the mirror database is known as the *mirror server*.

Witness: The optional *witness* is an SQL Server instance in a database mirroring configuration. It is separate from the principal and mirror instances. When database mirroring is used in synchronous mode, the witness provides a mechanism for automatic failover.

Send Queue: While sending the log records from the principal to the mirror, if the log records can't be sent at the rate at which they are generated, a queue builds up at the principal. This is known as the *send queue*. The send queue does not use extra storage or memory. It exists entirely in the transaction log of the principal. It refers to the part of the log that has not yet been sent to the mirror.

Redo Queue: While applying log records on the mirror, if the log records can't be applied at the rate at which they are received, a queue builds up at the mirror. This is known as the *redo queue*. Like the send queue, the redo queue does not use extra storage or memory. It exists entirely in the transaction log of the mirror. It refers to the part of the hardened log that remains to be applied to the mirror database to roll it forward.

Endpoint: An endpoint is a SQL Server object that enables SQL Server to communicate over the network. It encapsulates a transport protocol and a port number.

Failover: When the principal database (or the server hosting it) fails, database mirroring provides a mechanism to fail over to the mirror database.

Some important aspects of database mirroring are:

- The granularity of database mirroring is a database. Mirroring is configured for one database at a time. The whole instance is not mirrored.
- Two copies of the same database are involved in database mirroring, but only one database is accessible to the applications at any given time. You can create a snapshot on the mirror and use it for read-only purposes (a good solution for reporting requirements). However, you cannot directly access the mirror database or back up the mirror database.
- You cannot mirror the **master**, **msdb**, **temp**, or **model** databases.
- Database mirroring requires that the database use the full recovery model. You can't use the simple or bulk-logged recovery models.
- SQL Server 2005 allows only one mirror database for each principal database.
- One instance can serve as the principal for one database, the mirror for another database, and the witness for yet another database.
- Multiple databases in an instance can be mirrored.

- Applications that connect to a database with ADO.NET or the SQL Native Client (SNAC) can automatically redirect connections when the database fails over to the mirror.
- Data between the principal and the mirror is transferred encrypted by default.
- A database which is mirrored to another server can also be the source database for a log-shipping scenario.

Testing Methodology and Workloads

Microsoft performed a series of tests with various application profile workloads using a local area network (LAN), a metropolitan area network (MAN), and a wide area network (WAN) (both simulated and actual) to determine best practices for configuring database mirroring with synchronous and asynchronous operation. Database mirroring performance is a function of the application and environment characteristics. Application characteristics, such as the log generation rate, transaction commit rate, and number of concurrent database connections influence database mirroring performance. So do environment characteristics such as the transaction safety level, I/O throughput, and network performance.

Our hardware partner, NEC, provided the database servers and storage used in the testing discussed in this document. For details on the test environment, see Appendix A.

> **Note**: The results published in this document should not necessarily be treated as the optimum performance of the database mirroring feature on the described hardware. It is imperative that you validate your service level requirements by testing database mirroring with your application and hardware to determine the optimal configuration for your environment.

The tests were performed with two workloads. The workloads varied in database size, number of user connections, transaction rate, and redo generation rate. Both workloads included SELECT, INSERT, UPDATE, and DELETE operations. The application profiles for the two workloads are described in Table 1.

Characteristic	Workload1	Workload2
Database size (GB)	40	20
Number of concurrent user connections	1,000	20
Maximum think time between transactions (sec)	4	0
Baseline (no mirroring) %CPU	4	40
Data disk write rate (KB / sec)	4,500	1,500
Log disk write rate (KB / sec)	720	12,000
Baseline (no mirroring) transactions / sec	241	215
Baseline (no mirroring) log generation rate (KB / sec)	720	12,000

Table 1: Application profile for the workloads

Workload1 and Workload2 are both Online Transaction Processing (OLTP) applications. Workload1 is an order management system that tracks orders, payments, deliveries, and related information. It involves a high volume of random I/O, very low CPU usage, and a large number of concurrent users. Workload2 is a stock trading application. It captures stock quotes and orders. It involves a smaller database, but a higher log generation rate.

Of the characteristics shown in Table 1, the first three are measured or controlled through application settings—the rest are measured through Windows System Monitor (perfmon) counters. The most important application characteristic, with respect to database mirroring, is the log generation rate. You can measure the log generation rate of your application by monitoring the System Monitor counter **Log Bytes Flushed / sec** for the database. The amount part of the log that is generated by the application is the amount of information that is sent across the wire through the network to the mirror database, and therefore has the most significant impact on performance.

Transaction Safety Levels

One of the most important database mirroring configuration decisions is to select the appropriate transaction safety level. The transaction safety level determines whether the changes on the principal database are applied to the mirror database synchronously or asynchronously. SQL Server 2005 provides two safety levels—OFF and FULL.

Safety OFF is often referred to as *asynchronous mirroring.* Asynchronous mirroring sets up database mirroring with high performance. With safety OFF, the transaction is committed as soon as the principal server writes the log record to the local log and sends the log record to the mirror, without waiting for an acknowledgement from the mirror server. The database is synchronized after the mirror server catches up to the principal server. However, transactions commit without waiting for the mirror server to write the log record to the log file (also known as *hardening* the transaction log). If the principal can't send the log records to the mirror at the rate at which they are generated, a queue builds up at the principal. This is known as the send queue. (The send queue is monitored through the System Monitor counter **Log Send Queue KB** on the principal.) When the principal fails, some of the log records may not have been received by the mirror; because of this you may experience data loss. With asynchronous mirroring, you should notice only a negligible performance impact on the principal database.

Safety FULL is often referred to as *synchronous mirroring*. It provides high safety. Every transaction that is committed on the principal database is also committed on the mirror server synchronously, guaranteeing the safety of the data. This is achieved by having the principal server commit each transaction only after receiving an acknowledgement from the mirror server stating that it has hardened the transaction's log. As you might expect, there is a performance impact when you set the transaction safety level to FULL.

> **Note:** SQL Server 2005 Standard Edition allows only the FULL transaction safety level. The database mirroring features supported by various editions of SQL Server 2005 are described in the white paper Database mirroring in SQL Server 2005.

Table 2a compares the performance of Workload1 for safety OFF, safety FULL, and no mirroring.

Performance characteristic	No mirroring	Safety OFF	Safety FULL
% CPU	4.0	6.0	6.4
Data disk write rate (KB / sec)	4,500	4,500	4,500
Log disk write rate (KB / sec)	720	720	709
Transactions / sec	241	241	238
Transaction response time (seconds)	0.13	0.13	0.19

Table 2a: Performance characteristics of safety OFF and safety FULL for Workload1 (on the principal)

As illustrated in Table 2a, for Workload1, safety OFF has no impact on transaction throughput and response time. Setting the transaction safety level to FULL results in a slight reduction in transaction throughput and a slight increase in response time, compared to safety OFF. Table 2b illustrates the performance characteristics of the mirror server.

Performance characteristic	No mirroring	Safety OFF	Safety FULL
% CPU	N/A	3.9	4.0
Data disk write rate (KB / sec)	N/A	19,640	19,300
Log disk write rate (KB / sec)	N/A	720	709

Table 2b: Performance characteristics of safety OFF and safety FULL for Workload1 (on the mirror)

If you compare the disk write rates in Table 2a and Table 2b, you see that the mirror does the same amount of writes to the log disk as does the principal. However, the mirror does many more writes to the data disks than does the principal. This is because the data pages on the principal are written to disk only at the checkpoints whereas the pages on the mirror are written to disk continuously. The data pages on the mirror are aggressively queued for writes so that in the event of a failover there is not much to flush. The events that take place during failover are discussed in Failover Considerations in this paper.

The impact of the transaction safety level on performance is application dependent. For Workload2, with only 20 concurrent active database connections and a higher log generation rate than in Workload1, the performance impact is more substantial, as illustrated in Table 3a.

Performance characteristic	No mirroring	Safety OFF	Safety FULL
% CPU	39.6	47.4	37.7
Data disk write rate (KB / sec)	1,500	1,175	1217
Log disk write rate (KB / sec)	12,000	11,850	8720
Transactions / sec	215	211	158
Transaction response time (seconds)	6.4	6.7	8.8

Table 3a: Performance characteristics of safety OFF and safety FULL for Workload2 (on the principal)

For Workload2, safety OFF results in a 2% reduction in transaction throughput and adds 0.3 seconds to the response time compared to the no mirroring scenario. Turning on safety FULL, however, results in approximately a 25% reduction in transaction throughput and adds 2.4 seconds to the response time. Table 3b illustrates the performance characteristics of the mirror server.

Performance characteristic	No mirroring	Safety OFF	Safety FULL
% CPU	N/A	13.6	10.841
Data disk write rate (KB / sec)	N/A	37143	30036
Log disk write rate (KB / sec)	N/A	11856	8727

Table 3b: Performance characteristics of safety OFF and safety FULL for Workload2 (on the mirror)

Table 2a and Table 3a illustrate that the performance impact of the transaction safety level is highly dependent on the application. Applications involving a large number of concurrent connections experience less difference in throughput between safety OFF and safety FULL compared to applications involving a smaller number of concurrent connections. This is because, in the safety FULL mode with a large number of connections, it is possible that some connections can continue processing while other connections are waiting for acknowledgement from the mirror. This represents cases of Web farms that use connection pools to connect to SQL Server and some ISV applications such as SAP. With fewer connections, when some connections wait for acknowledgement from the mirror, the proportional impact on the overall transaction throughput is higher.

On similar lines, with safety FULL, short transactions experience more impact on response time than do long transactions. This is because, when transactions wait for acknowledgement from the mirror, the wait time adds proportionately more to the response time of the short transactions.

Impact of long and log-intensive transactions

Long and log-intensive transactions can impact performance and failover time. Some common examples of long and log-intensive transactions are creating or rebuilding an index on a large table and bulk loading a large amount of data.

Example 1: Index creation

Figure 1 illustrates the index creation time for a clustered and a nonclustered index for various mirroring safety levels. Creating an index on a large table takes longer with safety FULL compared to safety OFF. The performance impact of synchronous mirroring is more pronounced if you create a clustered index than if you create a nonclustered index. This is because creating a clustered index generates much more transaction log compared to that generated by a nonclustered index.

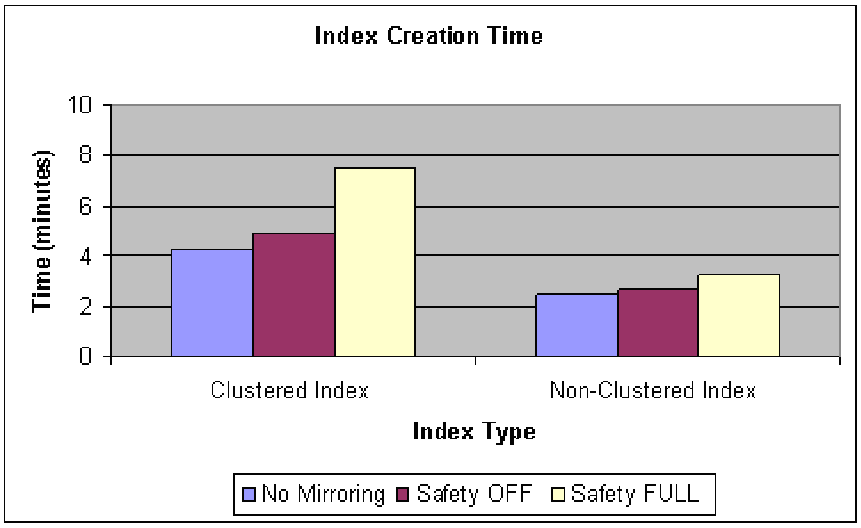


Figure 1: Index creation time with varying safety level

Example 2: Index rebuild

Rebuilding an index takes longer with safety FULL than it does with safety OFF. Synchronous mirroring introduces more overhead when rebuilding a clustered index than when rebuilding a nonclustered index. Figure 2 illustrates the index rebuilding time for a clustered and a nonclustered index for various safety levels. Figure 2 also compares the performance of ONLINE and OFFLINE index rebuilding for various safety levels. During an ONLINE index rebuild, the table and the index are available for queries and data modification, whereas during an OFFLINE index rebuild, a table lock is acquired and no other operation on the table is allowed.

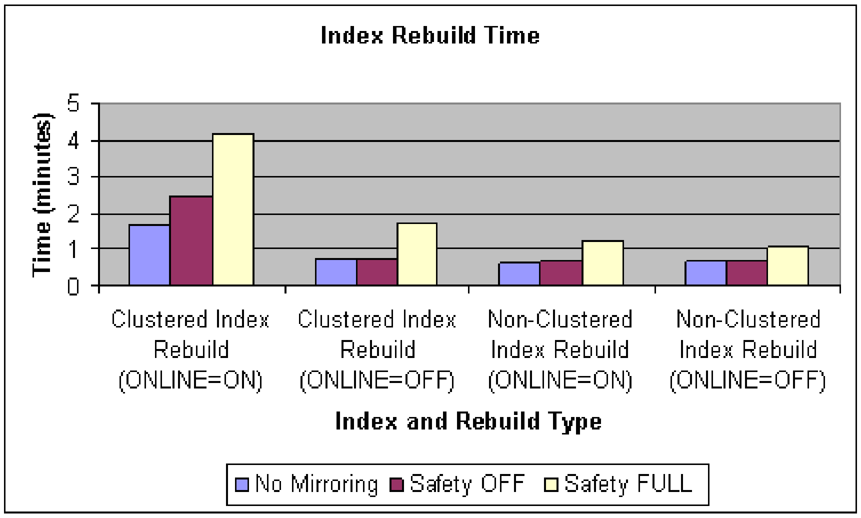


Figure 2: Index rebuild time with varying safety level

For long transactions, such as an index creation or rebuild, safety OFF seems to provide better performance compared to safety FULL. However, with safety OFF, the send queue on the principal and the redo queue on the mirror may build up significantly. Figure 3 illustrates the send queue and redo queue over time for the ONLINE rebuild of the clustered index scenario shown in Figure 2.

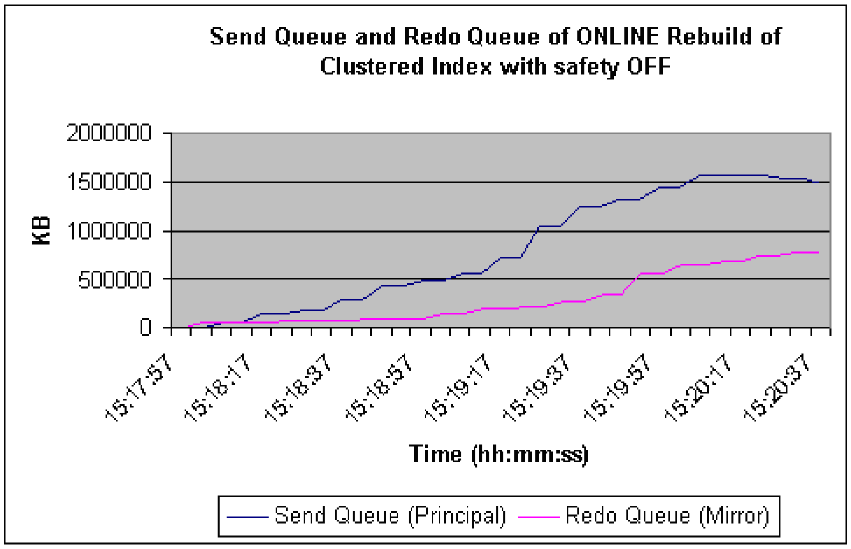


Figure 3: Send Queue and Redo Queue of ONLINE Rebuild of Clustered Index with safety OFF

As shown in Figure 3, the send queue and the redo queue build up constantly throughout the index rebuild process. After the rebuild is complete, it takes a significant amount of time for the mirror to synchronize. How long this takes depends upon the I/O and network performance on the principal and mirror. In our example, it took 3 minutes and 40 seconds for the send queue to come down to 0 after the index rebuild was complete; the redo queue takes much longer to subside.

Example 3: Bulk insert

We performed a bulk insert of 93 million rows into a table. We used varying batch sizes and varying safety levels. Bulk insert allows you to control how frequently you want to commit while inserting data. Batch size is the number of rows inserted before each commit; the batch size determines the size of each transaction. Our test involved batch sizes of 100, 1,000, and 1 million. The test involved loading data from a single data file into a single table. No parallel streams were involved. Figure 4 illustrates the performance of bulk insert with these batch sizes with no mirroring, safety OFF, and safety FULL.

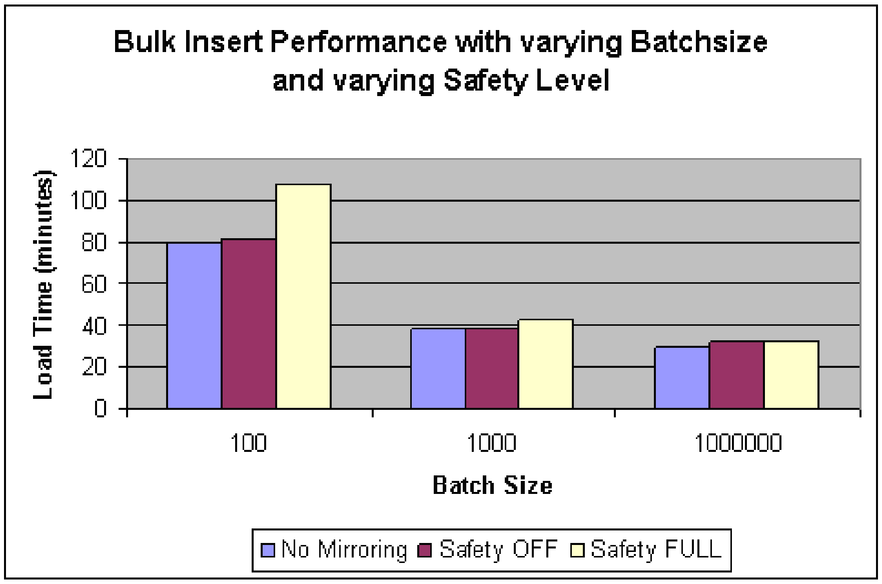


Figure 4: Bulk insert performance with varying batch size and safety level

As illustrated in Figure 4, the smaller the batch size, the longer it takes to complete the load. The more interesting observation is that for batch size = 100, the performance degradation caused by turning the safety level to FULL is more pronounced compared to larger batch sizes. For smaller batch sizes, the transaction size is smaller and the total number of transactions for a given data volume is higher. Each transaction experiences the overhead of waiting for the acknowledgement from the mirror. Therefore, the more frequent the commits, the higher the cumulative overhead for the complete load.

One important thing to note about the log bound workloads (workloads that generate a high volume of log traffic) such as Workload2, bulk insert, index creation, and index rebuild is that the performance under database mirroring is highly dependent upon the network performance and the log I/O performance. If either the network performance or the log I/O performance is a bottleneck, the performance of database mirroring for the workload can degrade significantly with the safety FULL operation. The potential for data loss may increase with the safety OFF operation if the network performance or the log I/O performance is a bottleneck.

Failover Considerations

Depending on the transaction safety level and whether you have a witness, you can have automatic failover, manual failover, or both. Table 4 lists the failover options for various safety levels.

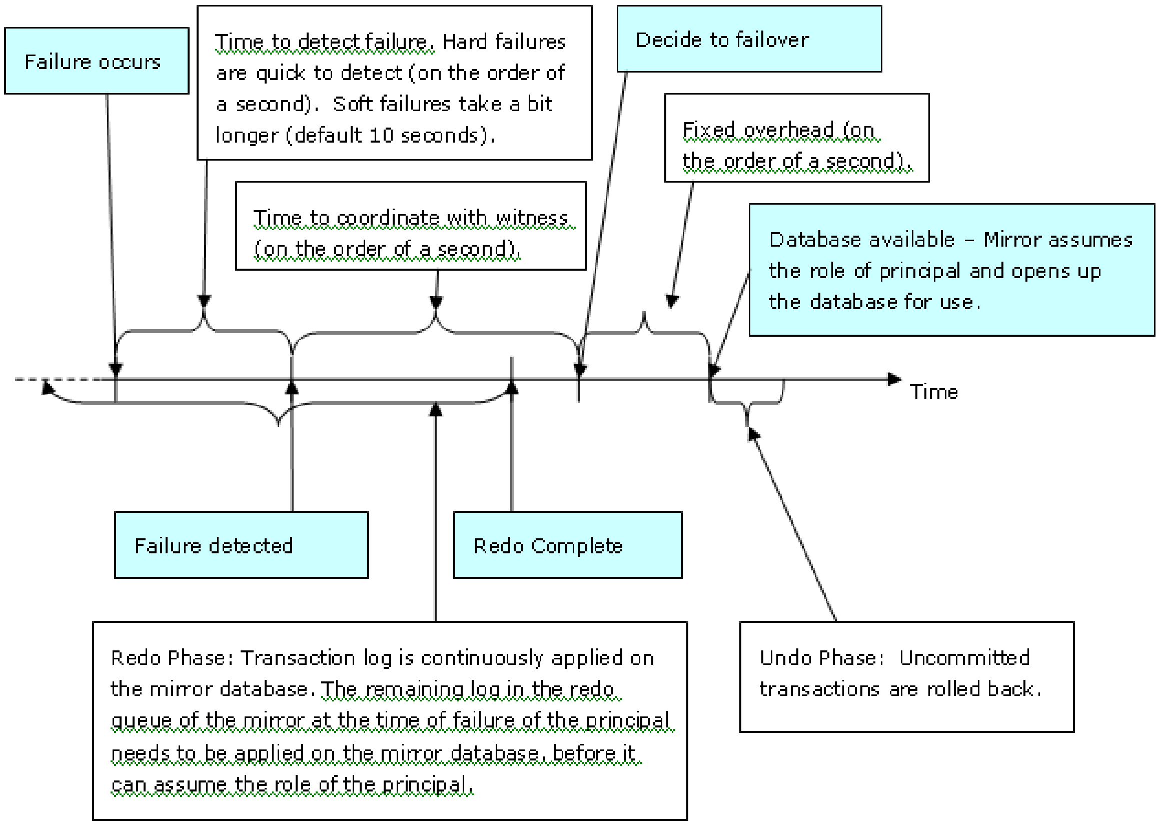
Transaction safety level	**Witness**	**Operating mode**	**Supported failover mode**
FULL	Yes	High safety with automatic failover (synchronous)	Automatic or Manual
FULL	No	High safety (synchronous)	Manual
OFF	N/A	High performance (asynchronous)	Forced Service (with possible data loss)

Table 4: Safety levels and failover options

When a failure occurs and automatic failover takes place, several events occur in the following sequence:

1. **Failure occurs**: The principal database becomes unavailable. This may be caused by hardware, storage, network, or power failure on the principal server.
2. **Failure is detected**: The failure is detected by the mirror and the witness. Hard failures (such as server power failure) are detected quickly (typically in around 1 second). Soft failures (such as storage failure) take a bit longer to detect. The default timeout for communication (a ping message) between the principal, the mirror, and the witness is 10 seconds. The timeout can be modified by using the ALTER DATABASE SET PARTNER TIMEOUT command. If the principal doesn't respond (to the ping message) within the timeout period, it is considered to be down. If you change the timeout setting, it is a best practices recommendation to set it to 10 seconds or higher. Setting it to lower than 10 seconds may result in false failures under heavily loaded or sporadic network conditions.
3. **Complete redo performed on mirror**: The mirror database has been in the Restoring state until now and the redo is being continuously applied on the mirror database. The remaining log in the redo queue of the mirror at the time of failure of the principal is applied on the mirror database to completely recover it.
4. **Decision to fail over**: The mirror coordinates with the witness and decides that the database should now fail over to the mirror. This decision process usually takes around 1 second. If the principal comes back before the redo phase in step 3 is complete, failover is not required.
5. **Mirror becomes principal**: After all the remaining redo is applied, the mirror database is brought online and it assumes the role of the principal. The database is now available and clients can connect to the new principal and continue operation.
6. **Undo**: The uncommitted transactions in the transaction log are rolled back.

Figure 5 illustrates the events that take place during automatic failover.


Failure occurs
Time to detect failure. Hard failures are quick to detect (on the order of a second). Soft failures take a bit longer (default 10 seconds).
Decide to failover
Fixed overhead (on the order of a second).
Time to coordinate with witness (on the order of a second).
Database available – Mirror assumes the role of principal and opens up the database for use.
Time
Failure detected
Redo Complete
Redo Phase: Transaction log is continuously applied on the mirror database. The remaining log in the redo queue of the mirror at the time of failure of the principal needs to be applied on the mirror database, before it can assume the role of the principal.
Undo Phase: Uncommitted transactions are rolled back.

The time from detecting the failure of the principal to the time the mirror assumes the role of the principal is the database failover time.

Figure 5: Events during automatic failover

From the server perspective, the database failover time is the amount of time that it takes from the time the failure of the principal is detected to the time the mirror assumes the role of the principal. A major component of the database failover time can be the redo phase. During this phase, the log in the redo queue of the mirror is applied to the mirror database (in some cases, the mirror may already be caught up with the principal, and the redo phase will not introduce a time lag). The time to apply the redo records depends on the redo queue length and the redo rate on the mirror. These are measured by the System Monitor counters **Redo Queue KB** and **Redo Bytes/sec**, respectively. You can divide the redo queue by the redo rate at any given time to estimate the time it will take to apply the log in the redo queue of the mirror. SQL Server Books Online describes the redo apply and failover time estimation in detail.

One way to measure the actual failover time is to use SQL Server Profiler on the principal and the mirror. In SQL Server Profiler, you create a trace and select the **Database Mirroring State Change** event to monitor. You should look at two columns—**StartTime** and **TextData**. **StartTime** is a timestamp representing the time at which the event takes place. **TextData** is the description of the database mirroring state change event. When the mirror server changes its state to principal, SQL Server Profiler outputs messages in the trace. Table 5 lists the **StartTime** and **Textdata** for one example failover scenario.

StartTime	TextData
2005-10-22 12:39:17.960	DBM: Synchronized Mirror with Witness -> DBM: Connection with Principal Lost
2005-10-22 12:39:18.570	DBM: Connection with Principal Lost -> DBM: Automatic Failover
2005-10-22 12:39:20.590	DBM: Automatic Failover -> DBM: Principal Running Exposed

Table 5: Event timestamp and description from the trace

The length of the failover time depends upon the type of failure and the load on the database. Failover time was measured for different types of failures and failovers under no load and normal load conditions. To measure the failover time under load Workload1 was executed for one hour. Then the failure scenario was created. Some sample failover times are listed in Table 6.

Failure / Failover type	Failover time with no load (sec)	Failover time with normal load (sec)
ALTER DATABASE FAILOVER (manual failover)	20.5	32.2
SHUTDOWN WITH NOWAIT	5.7	8.5
Stop SQL Server Service on principal	3.2	10.8
Shutdown principal server	10.2	15.8
Power OFF principal server	2.2	2.6
Pull network cable on principal connecting to mirror and witness	1.9	3.1

Table 6: Failover time (synchronous mirroring with witness) for Workload1

As illustrated in Table 6, for a given failure type, it takes longer to complete the failover under load as compared to the no-load scenario. This is because once the failure is detected, the mirror must apply the transactions in the redo queue before it can assume the role of the principal and make the database available.

Manual failover takes longer than automatic failover. During manual failover, you see messages similar to the following in the ERRORLOG of the mirror (which eventually becomes the principal after the failover completes):

```
Starting up database '<db_name>'.

Analysis of database '<db_name>' (<dbid>) is 100% complete (approximately
0 seconds remain).

Recovery of database '<db_name>' (<dbid>) is 100% complete (approximately
0 seconds remain). Phase 3 of 3.
```

These additional steps during analysis and recovery cause the manual failover to take longer.

The following sections discuss various failure scenarios and their effects.

Loss of the principal

If the principal fails, the failover scenario depends on the transaction safety level and whether you have a witness.

Scenario 1: Safety FULL with a witness

This scenario provides the high safety with automatic failover. In the event of the failure of the principal, the mirror forms a quorum with the witness. Automatic failover will take place, thereby minimizing the database downtime.

For example, prior to the failure, Server_A and Server_B acted as principal and mirror, respectively. Server_A fails. Following the automatic failover, Server_B takes on the role of the principal. However, since there is no mirror after the failover, the mirroring state is DISCONNECTED and the principal is exposed. Once the database on Server_A becomes operational, it automatically assumes the role of the mirror.

Scenario 2: Safety FULL without a witness

This scenario provides high safety, but automatic failover is not allowed. In the event of failure of the principal, the database service becomes unavailable. You need manual intervention to make the database service available. You must break the mirroring session and then recover the mirror database.

For example, prior to the failure, Server_A and Server_B acted as principal and mirror respectively. Server_A fails. You need to execute the following on Server_B to make the database service available:

```
ALTER DATABASE <database name> SET PARTNER OFF

RESTORE DATABASE <database name> WITH RECOVERY
```

Once Server_A becomes available, you need to re-establish the mirroring session.

Scenario 3: Safety OFF

If the safety level is OFF, the witness doesn't add any value to the database mirroring scenario. Therefore, it is recommended that if you plan to run database mirroring when the safety level is OFF, don't configure a witness. In the event of failure of the principal, the database service becomes unavailable. You can perform a force service to make the database service available on the mirror. However, since the safety level is OFF, it is possible that there were transactions that didn't make it to the mirror at the time of the failure of the principal. These transactions will be lost. Therefore, manual failover with safety OFF involves acknowledging the possibility of data loss.

For example, prior to the failure, Server_A and Server_B acted as principal and mirror respectively. Server_A fails. You need to execute the following on Server_B to make the database service available:

```
ALTER DATABASE <database_name> SET PARTNER FORCE_SERVICE_ALLOW_DATA_LOSS
```

Once the database on Server_A becomes operational, it automatically assumes the role of the mirror. However, the mirroring session remains SUSPENDED, and you will need to manually RESUME the mirroring session.

Another option in the event of failure of the principal, is to break the mirroring session and then recover the mirror database, as in Scenario 2: Safety FULL without a witness. Whether you choose to break the mirroring session or force service, you will lose the transactions that haven't yet made it to the mirror at the time of failure.

Loss of the mirror

If the mirror fails, the principal continues functioning, but the mirroring state is DISCONNECTED and the principal is running exposed. Once the mirror database becomes operational, it automatically assumes the role of the mirror and starts synchronizing with the principal. For as long as the mirroring state stays DISCONNECTED, the transaction log space on the principal cannot be reused, even if you back up the transaction log. If the log file grows and reaches its maximum size limit or runs out of disk space, the complete database comes to a halt. To prevent this you have two options—either plan for enough disk space for the transaction log to grow and bring back the mirror database before the space fills up, or break the database mirroring session.

Loss of the witness

If the witness server fails, database mirroring continues functioning without interruption, except that automatic failover is not possible. Once the witness becomes operational, it automatically joins the database mirroring session.

Loss of the mirror and the witness

Assume you have configured database mirroring with a witness. When the mirror is unavailable, the principal runs exposed. While the mirror is unavailable, if the witness is also lost, the principal becomes isolated and can't service the clients. Even though the principal database is running, it is not available to the clients. If you attempt to connect to the database, you get the message "Database <dbname> is enabled for database mirroring, but neither the partner nor witness server instances are available: the database cannot be opened."

If the mirror or the witness cannot be brought back online quickly, then the only way to resume database service is to terminate the database mirroring session. To do this, you execute the following command after connecting to the master database of the principal server:

```
ALTER DATABASE <db_name> SET PARTNER OFF
```

Note that you need to execute a SET PARTNER OFF command, and not a SET WITNESS OFF. SET WITNESS OFF will not work in this situation.

Once the mirror becomes available, you can re-establish the database mirroring session. The principal will start sending the log information to the mirror, and the mirror will eventually catch up. If you backed up the transaction log after terminating the database mirroring session, you need to restore the transaction log backup on the mirror before you can re-establish the database mirroring session. There is no need to do a full database backup / restore. Once the witness becomes available, you can join in the witness as well, but you have to establish the mirroring session with the mirror before the witness can join in.

Database availability during planned downtime

If you configured database mirroring with safety FULL, you can switch to the mirror server and make it available to the clients while you perform hardware or software maintenance on the principal. This "rolling upgrade" technique can be used to perform hardware or software changes on the principal and the mirror servers as follows:

1. Perform the hardware or software change on the mirror server first. If you need to restart the SQL Server service on the mirror or restart the mirror server, you can do so. As soon as the mirror database comes back up, the mirroring session is re-established automatically and the mirror starts synchronizing itself against the principal. (Note that the principal is exposed for the duration for which the mirror database is down, or up but not yet fully synchronized with the principal.)
2. Once the mirroring state becomes **SYNCHRONIZED**, switch the principal and mirror (by executing the **ALTER DATABASE FAILOVER** command). Now the applications can connect to the new principal database and continue service. Open or in-flight transactions will be rolled back while switching the principal and mirror. Depending on the type of application, it is recommended that you stop the application for the brief moment of manual failover and restart after the failover succeeds. In this case, please note that you will need to point your application to the new principal database server. You can now perform the hardware and software changes on the old principal server. As soon as you are done with the change and the database becomes available, it automatically re-establishes the mirroring session and assumes the role of the mirror.
3. If you have a witness server, you can now perform the required changes on that server.
4. You have two options—either leave the principal and mirror alone, or switch the principal and mirror (by executing the **ALTER DATABASE FAILOVER** command) after the mirroring state becomes **SYNCHRONIZED**. Assuming the mirror has the same capacity as the principal, it is recommended that you leave the database there since switching the principal and mirror may cause another reconnect for the application.

If you have configured database mirroring with safety OFF, you can still use the rolling upgrade technique after switching to safety FULL. During an off-peak period, you can turn the safety level to FULL and wait for the mirror to catch up and be synchronized with the principal. Now, you can perform manual failover during the planned downtime without losing any data. Note that if you switch from safety OFF to safety FULL during a peak load period, it may take a very long time for the mirror to catch up and be synchronized with the principal. You can't perform manual failover until it is synchronized.

Note: The rolling upgrade technique works in most hardware and software upgrades. However, there may be exceptions. It is a best practices recommendation that you test the rolling upgrade in a test environment before performing it in the production environment.

Suspending and resuming a mirroring session

You can suspend a mirroring session, and later resume it. To suspend a database mirroring session, execute:

```
ALTER DATABASE <db_name> SET PARTNER SUSPEND
```

To resume the database mirroring session, execute:

```
ALTER DATABASE <db_name> SET PARTNER RESUME
```

During the time the session is suspended, log records keep accumulating at the principal. The transaction logs on the principal cannot be truncated during the suspended period (even if you back up the transaction log). If you keep the mirroring session suspended for an extended period of time, you may fill up the log space and the database will come to a halt. Therefore, if you want to keep the mirroring session suspended for longer than a few minutes, you should plan for enough room for the transaction log to grow, depending upon your log generation rate.

The transaction log on the principal can also keep growing if the mirror database remains unavailable for an extended period of time. When the mirror database becomes unavailable, the database mirroring session remains DISCONNECTED, and the transaction log on the principal can't be truncated during the DISCONNECTED period (even if you back up the transaction log).

In one experiment we suspended the mirroring session for an extended period of time. During this time several bulk loads were executed. These filled up the transaction log. The transaction log automatically extended itself and ultimately the disk volume filled up. Since there was no space left for the transaction log to grow, the database came to a halt. The high-volume data loads filled up the 40-GB disk space for the transaction log in slightly over two hours.

If truncation and reuse of the transaction log space is prevented because the database mirroring state is either **SUSPENDED** or **DISCONNECTED**, the **LOG_REUSE_WAIT_DESC** column in the **sys.databases** catalog view will have the value **DATABASE_MIRRORING** for that database. In this situation, you have two options. You can configure enough space for the transaction log to grow (but, you will hit the space limit sooner or later, unless you **RESUME** the mirroring session or bring back the mirror database soon). Or, you can break the database mirroring session by executing the following command:

```
ALTER DATABASE <db_name> SET PARTNER OFF
```

Security Considerations

Database mirroring involves the transfer of information from one computer to another over a network. Therefore, the security of the information transfer is of great importance. By default, database mirroring transfers information in encrypted format. For configuring database mirroring between untrusted network domains, certificates are needed to authenticate connections.

Endpoint encryption

Connection management for database mirroring is based on endpoints. An endpoint is a SQL Server object that enables SQL Server to communicate over the network. For database mirroring, a server instance requires a dedicated database mirroring endpoint to receive database mirroring connections from other server instances. An endpoint that is used for a database mirroring connection cannot be used for any other purpose. Moreover, each SQL Server instance can have only one database mirroring endpoint. All databases in that instance that are involved in database mirroring must use the same endpoint.

Database mirroring endpoints use Transmission Control Protocol (TCP) to send and receive messages between the server instances in database mirroring sessions. The database mirroring endpoint of a server instance is associated with the port on which the instance listens for database mirroring messages. Each database mirroring endpoint listens on a unique TCP port number.

By default, a database mirroring endpoint requires the encryption of data that is sent over mirroring connections. In this case, the endpoint can connect only to endpoints that also use encryption. Unless you can guarantee that your network is secure, we recommend that you require encryption for your database mirroring connections. However, if you have a guaranteed secured network, you can turn off encryption, and gain a bit of performance.

Database mirroring endpoints support two encryption algorithms—RC4 and AES. RC4 is the default. Figure 6 illustrates the impact of endpoint encryption (with RC4 algorithm) on transaction throughput for Workload1.

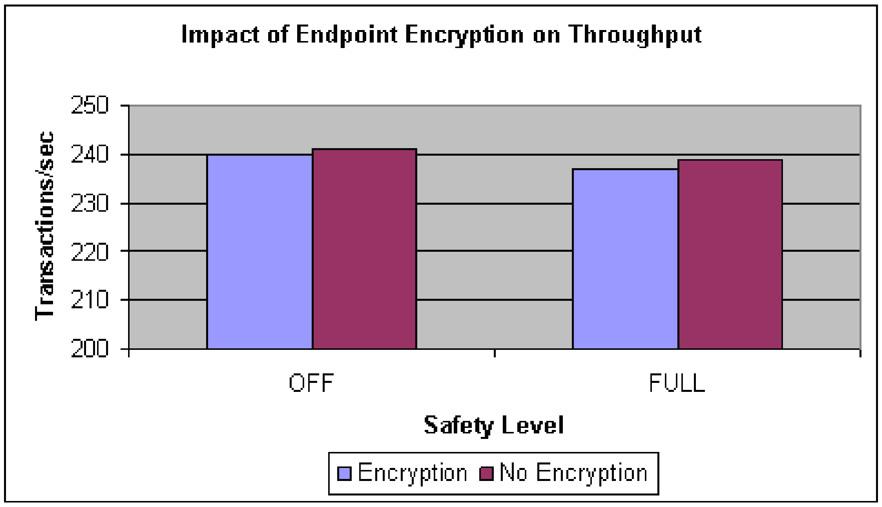


Figure 6: Transaction throughput with endpoint encryption with RC4 algorithm

Figure 7 illustrates the impact of endpoint encryption (with RC4 algorithm) on response time for Workload1.

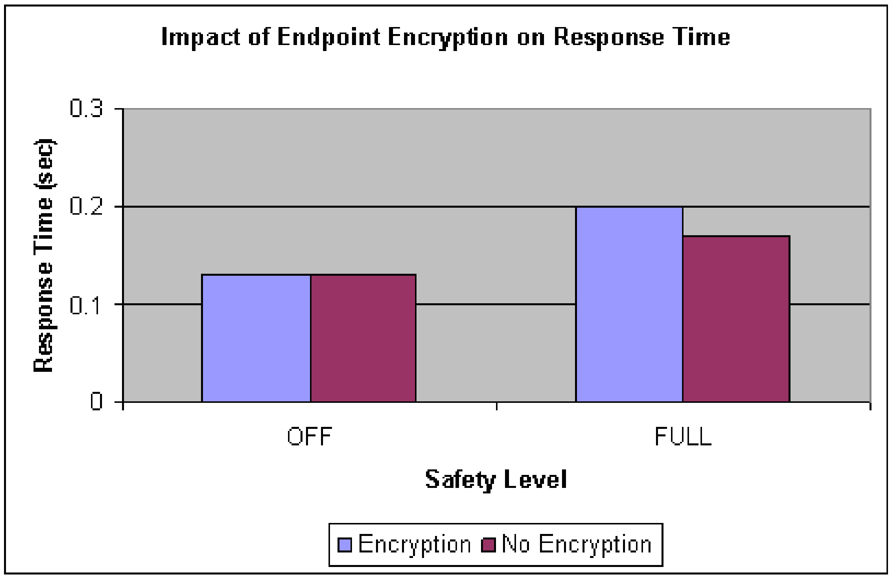


Figure 7: Transaction response time with endpoint encryption with RC4 algorithm

The numbers in Figure 6 and Figure 7 show a marginal impact on performance. Therefore, it is always a best practice to configure database mirroring endpoints with encryption.

When you create a database mirroring endpoint by using the CREATE ENDPOINT statement, the endpoint is created with encryption by default. If you decide to change the encryption of an endpoint after establishing mirroring, you need to take some extra care. The CREATE or ALTER ENDPOINT statements allow three possible values for the ENCRYPTION argument—REQUIRED, SUPPORTED, or DISABLED. (REQUIRED is the default.) For database mirroring to work, the endpoints of the principal, the mirror, and the witness (if there is one) all must be compatible. An endpoint with ENCRYPTION=REQUIRED cannot communicate with an endpoint with ENCRYPTION=DISABLED and vice versa. Changing the encryption of one endpoint from DISABLED to REQUIRED (or vice versa) breaks the mirroring session. Therefore, you need to change the encryption settings of the database mirroring endpoints from DISABLED to REQUIRED in two steps. First change the encryption from DISABLED to SUPPORTED for all the endpoints involved in the mirroring session (principal, mirror and witness, if any). Then change the encryption from SUPPORTED to REQUIRED for all the endpoints. The middle state of SUPPORTED allows the endpoint to communicate with other endpoints with either REQUIRED or DISABLED encryption.

Certificates

We tested database mirroring between two remote database servers—one in Redmond, Washington, and the other in Charlotte, North Carolina. The two database servers are in different and untrusted domains. To establish database mirroring between untrusted domains, certificates are required to authenticate connection requests. Certificates have no impact on performance and provide a secured way of authenticating database mirroring connections. On each server (principal, mirror, and witness), the endpoint is configured with a locally created certificate. Then the certificates are copied to the other servers.

While copying certificates and database backups between servers in different domains, you should always use a secured copy method, such as winscp.

For details on how to create certificates and how to configure database mirroring using certificates, see SQL Server Books Online.

Network Configuration Best Practices

Network configuration plays an important role in the performance and safety provided by database mirroring. Extensive tests with database mirroring were performed on a local area network, and a simulated wide area network, and a metropolitan area network to determine the impact of network latency (roundtrip times) and network bandwidth on the throughput of the principal database. Table 7 lists the roundtrip times (RTT) typically observed in local, metropolitan, and wide area networks.

Network type	**Typical roundtrip times (RTT) in milliseconds (ms)**
Local area network	0 to 2
Metropolitan area network	2 to 20
Wide area network	20 to 200

Table 7: Type of network and associated network latency

Database mirroring on a local area network

On a local area network, database mirroring can help minimize unplanned as well as planned downtime. Synchronous database mirroring with a witness is a viable option in a local area network, since the network latency is usually low, thereby offering the benefits of high availability to reduce unplanned downtime.

Database mirroring can also be used to minimize planned downtime due to hardware upgrades, software upgrades, storage changes, and database configuration changes as described in Database Availability During Planned Downtime earlier in this white paper.

Within a reliable LAN, synchronous mirroring with a witness is recommended if availability is of higher importance than performance. However, if performance is of paramount importance, asynchronous mirroring is preferred. You need to make this design tradeoff based on the performance and availability requirements of your application and the performance impact of synchronous mirroring on your application.

Database mirroring on a metropolitan or wide area network

On a metropolitan or wide area network, database mirroring is often used as a disaster-recovery solution. In a typical metropolitan area network, roundtrip times (RTT) range between 2 milliseconds (ms) to 20 ms, whereas in a wide area network, the RTT can be as long as 200 ms. Database mirroring can be set up in a MAN or WAN environment to provide the means to maximize performance or minimize data loss.

Figure 8 illustrates the variation of transaction throughput and response time with increasing network latency in a synchronous mirroring scenario for Workload1.

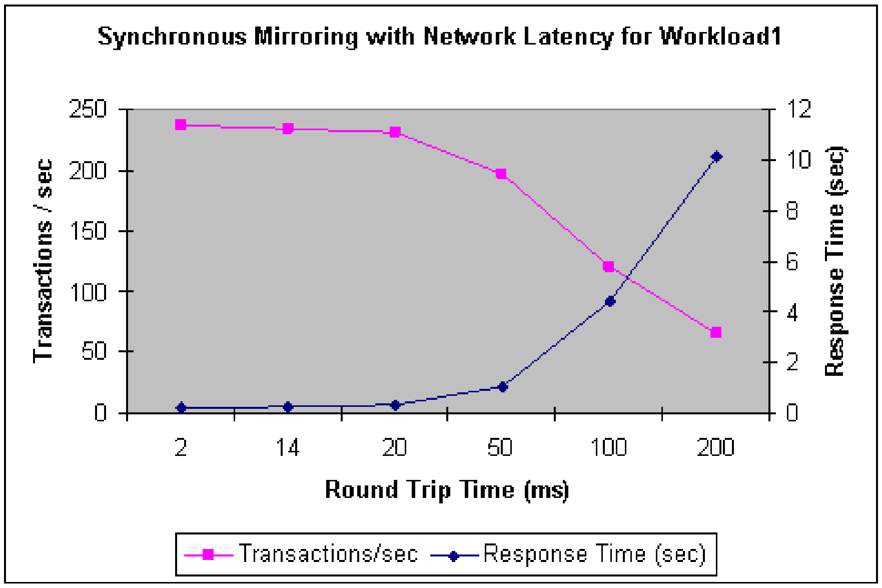


Figure 8: Synchronous mirroring performance with increasing network RTT for Workload1

As Figure 8 illustrates, there is very minimal change in transaction throughput and response time for RTT values below 20 ms. Performance starts degrading for RTT values greater than 20 ms.

Figure 9 illustrates the variation of transaction throughput and response time with varied network latency in a synchronous mirroring scenario for Workload2.

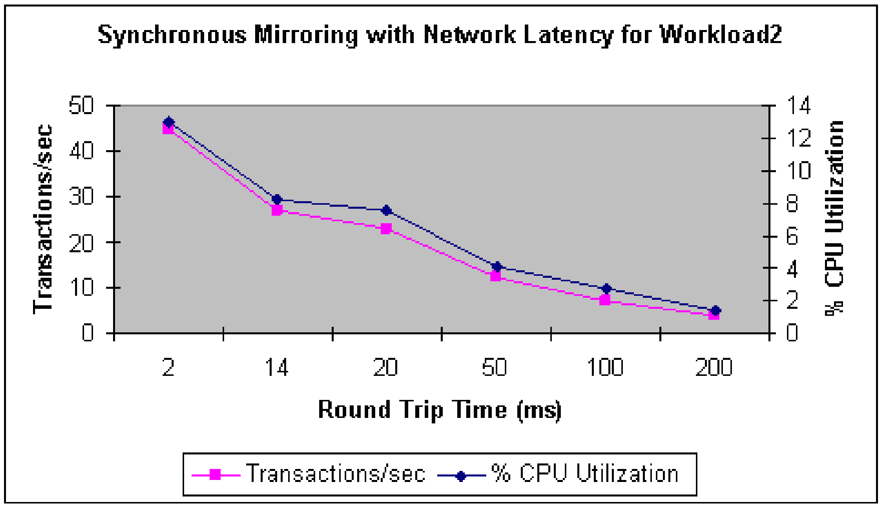


Figure 9: Synchronous mirroring performance with increasing network RTT for Workload2

As Figure 9 illustrates, transaction throughput and % CPU Utilization steadily degrade as RTT increases. If the network latency between the principal and mirror servers is 2 ms or longer, evaluate the observed performance with respect to the performance goals of your application. Then you can determine if synchronous mirroring is the right choice in the given environment.

Figure 10 illustrates the variation of transaction throughput and response time with increasing network latency in an asynchronous mirroring scenario for Workload1.

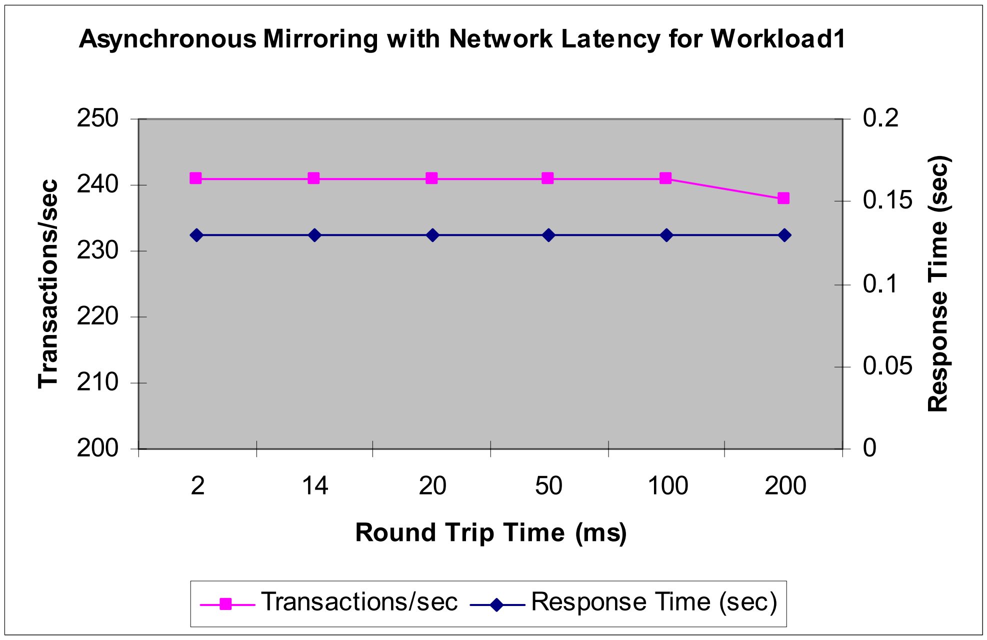


Figure 10: Asynchronous mirroring performance with increasing network RTT for Workload1

As Figure 10 illustrates, transaction throughput remains the same for RTT values up to 100 ms and degrades only marginally for RTT values > 100 ms. For this workload, the response time remains the same throughout. However, with high network latency, the log send queue can build up significantly under heavy workload, which results in longer failover time and increased transaction loss in the event of a failover. For Workload1 with asynchronous mirroring, the log send queue remains low for RTT values up to 50 ms.

However, the queue builds up increasingly for RTT values of 100 ms or more. Figure 11 shows the log send queue for workload1 with RTT = 100 ms for the asynchronous mirroring scenario.

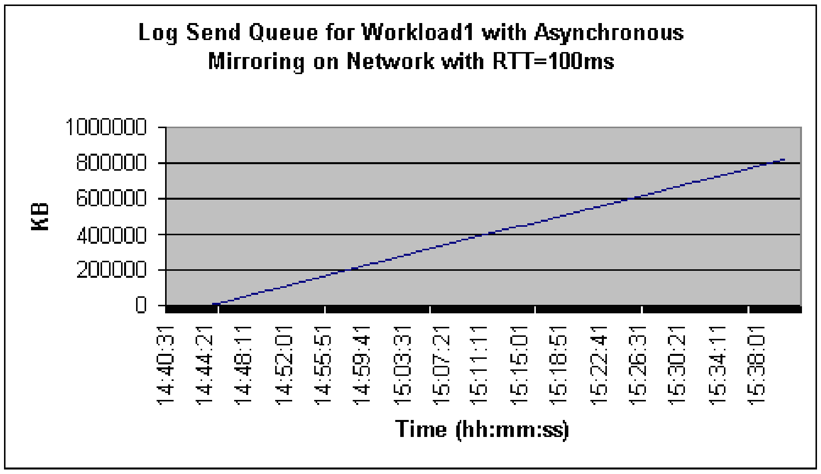


Figure 11: Log send queue for Workload1 with asynchronous mirroring on network with RTT=100ms

As illustrated in Figure 11, the log send queue builds up increasingly for Workload1 with asynchronous mirroring on a network with 100 ms RTT. Therefore, you should verify performance along with the log send queue while deploying asynchronous mirroring on high-latency networks.

Figure 12 illustrates the variation of transaction throughput and response time with varied network latency in an asynchronous mirroring scenario for Workload2.

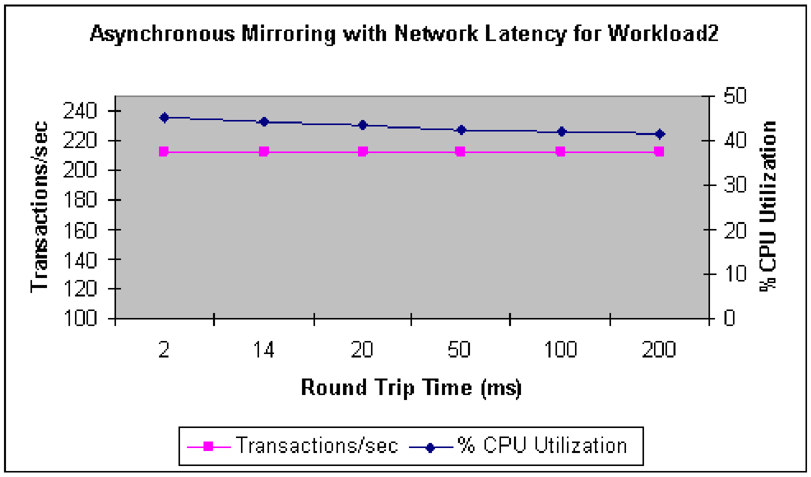


Figure 12: Asynchronous mirroring performance with varied network RTT for Workload2

As Figure 12 illustrates, transaction throughput remains unaffected with an increase in network latency. However, for this workload, unlike Workload1, the log send queue builds up increasingly even for RTT values as low as 2 ms.

Database mirroring on low-bandwidth networks

Ideally, you should have a dedicated network between the servers that are involved in database mirroring. If the servers share a common network with other servers, the effective bandwidth that is available for mirroring communication will impact performance. It is highly recommended that you configure database mirroring in a high-bandwidth network. The network bandwidth is dictated by the log generation rate of your application. Lower bandwidth networks can adversely impact the performance of database mirroring.

Figure 13 illustrates the variation of transaction throughput and response time with respect to varying the network bandwidth for Workload1 with synchronous mirroring.

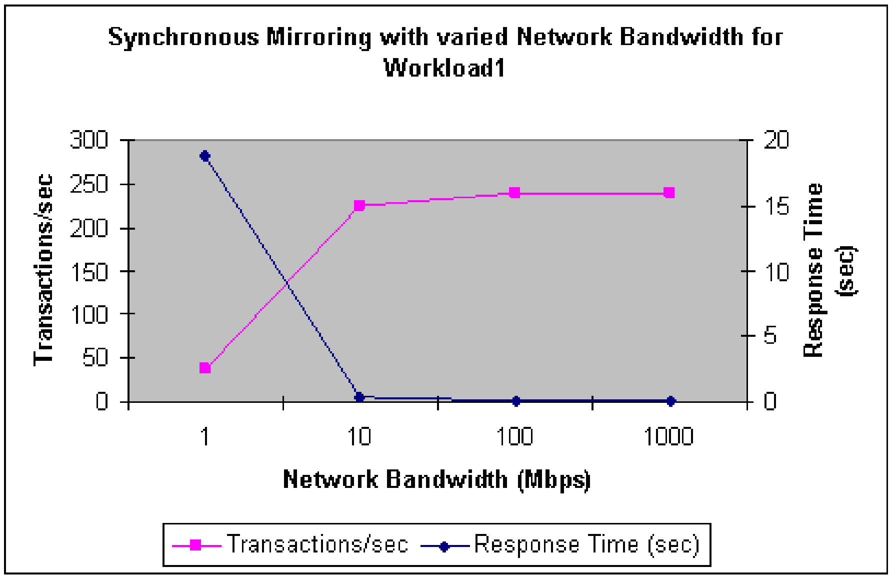


Figure 13: Impact of network bandwidth on synchronous mirroring with Workload1

As illustrated in Figure 13, good transaction throughput and response time is maintained for networks with bandwidth values of 10 megabits per second (Mbps) and higher. For network bandwidth less than 10 Mbps, throughput and response time degrade drastically. (We didn't attempt to measure performance with synchronous mirroring in a 56-kilobits per second (Kbps) network as it was extremely slow. However, we did this for asynchronous mirroring.)

The key thing to note in Figure 13 is that this application generates transaction log at the rate of 720 KB/sec (see Table 1), which is approximately 5.6 Mbps. This amount of transaction log needs to be transferred to the mirror server over the network. Therefore, network bandwidth must be significantly more than the log generation rate of the application. Otherwise, network performance will be a major bottleneck in synchronous mirroring performance. This is illustrated by the sudden downgrade in transaction throughput and response time for bandwidth = 1 Mbps in Figure 13.

Figure 14 illustrates the variation of transaction throughput and response time with respect to varying the network bandwidth for Workload1 with asynchronous mirroring.

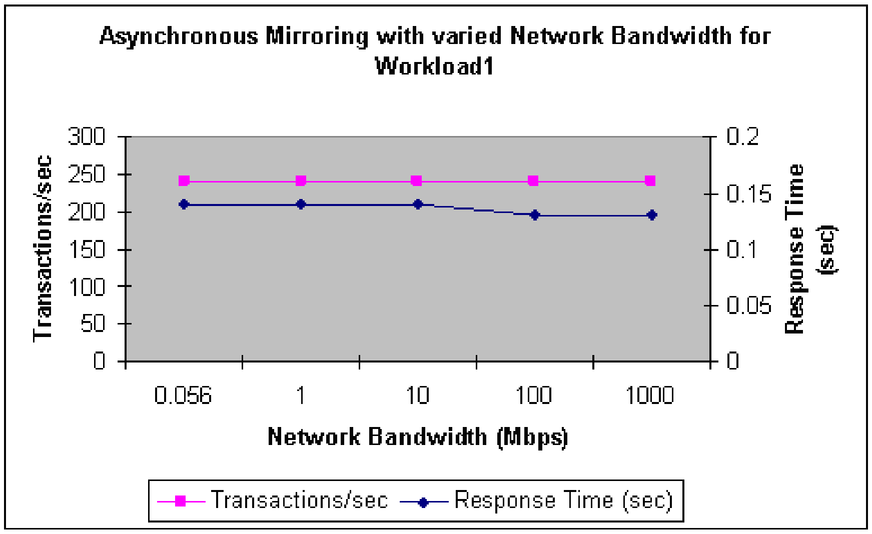


Figure 14: Impact of network bandwidth on asynchronous mirroring with Workload1

As illustrated in Figure 14, transaction throughput remains the same for high- as well as low-bandwidth networks. Response time increases only marginally for networks with a bandwidth less than 100 Mbps. Therefore, asynchronous mirroring is possible with low-bandwidth networks. However, keep in mind that for asynchronous mirroring on low-bandwidth networks, under heavy transaction loads, the log send queue can build up significantly. This can result in the loss of a significant amount of data if the principal database fails.

Figure 15 shows the log send queue for Workload1 with network bandwidth = 1 Mbps for the asynchronous mirroring scenario.

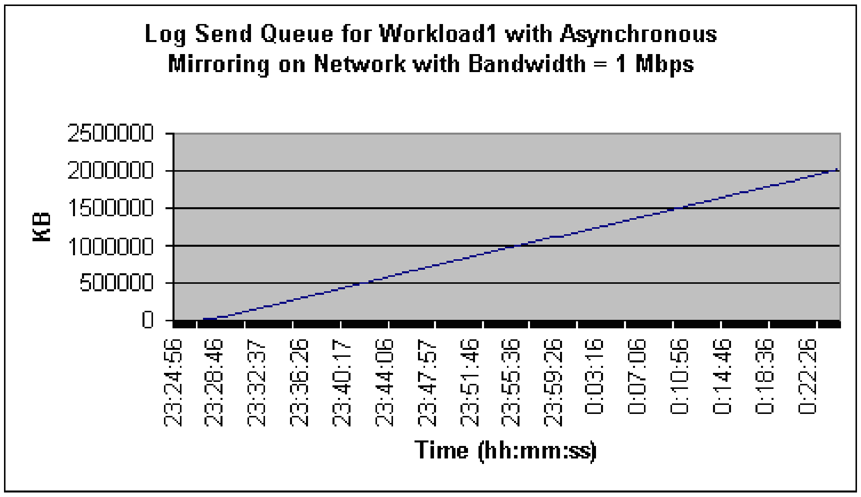


Figure 15: Log Send Queue for Workload1 with Asynchronous Mirroring on Network with Bandwidth = 1 Mbps

As illustrated in Figure 15, the log send queue builds up indefinitely for Workload1 with asynchronous mirroring on a network with 1 Mbps bandwidth. Therefore, you should verify both performance and the log send queue when deploying asynchronous mirroring on low-bandwidth networks.

The send queue at a given time represents the amount of data that would be lost if a principal failure were to occur at that time. With asynchronous mirroring, the amount of data loss that can be tolerated determines the optimum network latency and bandwidth requirements.

Preparing the Mirror for Failover

In the event of a failover, you want the mirror database to take up the full workload of the principal database. Therefore, it makes sense to use identical servers (in terms of CPU, memory, storage, and network capacity) as principal and mirror partners. Best practices recommendations are to use:

- Identical partner servers (in terms of CPU, memory, storage, network capacity).
- The same service pack and patch levels for the operating system and SQL Server on both partners. When performing rolling upgrades, the service pack and patch levels can be temporarily different on the principal and the mirror. However, for steady state operations, they should be identical.
- The same edition of SQL Server on both partners.
- An identical directory structure for the SQL Server install and database files on both partners.
- The same SQL Server configuration (trace flags, startup options, memory settings, etc.) for the principal and mirror instances.

If the principal and mirror servers are not identical in terms of CPU, memory, storage, and network capacity, you might experience undesirable performance when you fail over. Using the same edition of SQL Server, and the same version and service pack level of SQL Server and the operating system on both the partners minimizes the risks of encountering SQL Server or operating system issues. Ensuring that both servers have the same directory structure and SQL Server configuration eliminates the need for changes to these during or after the failover to the mirror partner.

The granularity of database mirroring is a single database. Each database in an instance of SQL Server is mirrored independently. The entire instance is not mirrored. For full instance failover capabilities (not database level), consider failover clustering. (For details on failover clustering, see SQL Server 2005 Books Online.). Moreover, only user databases can be mirrored. You cannot mirror the **master**, **msdb**, **tempdb**, or **model** databases. Therefore, you need to perform some administrative tasks to prepare the mirror server to take over the role of principal in the event of a failover. Several considerations apply to the mirror server:

- Make sure that applications can connect and execute all necessary actions, and that all active SQL Server logins (and their permissions) on the principal server are also present on the mirror server. You can use the Transfer Logins task of SQL Server 2005 Integration Services to accomplish this. (For details on the Transfer Logins task, see SQL Server 2005 Books Online.)
- Copy the SQL Agent jobs, alerts, SSIS packages, support databases, linked server definitions, backup devices, maintenance plans, database mail profiles, etc. from the

principal server to the mirror server. The jobs on the mirror server are typically disabled. Make sure to enable them in the event of a failover.

- If you add a disk volume on the principal partner on which you plan to put data or log files for the database that is being mirrored, add a disk volume with the same drive letter on the mirror server as well. Otherwise, when you add a data file on the new drive on the principal, database mirroring tries to add the same file in the mirror. This will fail because the new drive doesn't exist on the mirror server.
- Have a process in place so that when you make any changes (such as changes to hardware, software, SQL Server settings, or any database support objects) to the principal, you repeat or transfer the changes on the mirror server.
- Test server failover and make sure that all objects, logins, permissions, etc. work on the mirror the same as on the principal.

Initial Synchronization Between Principal and Mirror

Prior to establishing the database mirroring session, the database must be restored on the mirror server, and be in the Restoring state. The most common way to achieve this is to take an online backup of the database on the principal, copy the backup to the mirror server, and restore it on the mirror server with the option to apply further transaction logs.

Depending on the size of the database and the distance between the two servers, it can take a long time to copy the backup and restore it on the mirror server. During this time, it is possible that the principal database has produced several transaction log backups. If the principal produces lots of transaction log backups, you need a way to copy all the transaction log backups and apply them on the mirror. If you don't want to bother copying and applying so many transaction log backups on the mirror, you can suspend transaction log backups on the principal until the database on the mirror server is restored and the database mirroring session is established. After the database mirroring session is established, resume transaction log backups on the principal. One thing to note in this approach is that during the time that transaction log backups are suspended, the log file can grow; you need to plan for enough disk space to account for the growth of the log file.

Mirroring Multiple Databases

If there are multiple databases in an instance, you need to mirror each database individually. Although it is possible to choose a different mirror server for each database, it is a best practice recommendation that you choose one mirror server for all the databases that belong to one application. If the databases in an instance belong to different applications, you can choose different mirror servers for them, but doing so adds complexity to the configuration.

One concern arises when multiple databases are involved. What if one database fails over and the other doesn't? For example, Server_A is the principal server and Server_B is the mirror server for two user databases, DB1 and DB2. If a failover occurs with database DB1 and not with database DB2, then Server_B will be the principal for DB1 and Server_A will be the principal for DB2. If databases DB1 and DB2 belong to two different applications, then one application can continue connecting to Server_A, while the other reconnects to Server_B.

However, if both DB1 and DB2 belong to one application, which server does the application connect to? Even though both databases are available, the application's service might be unavailable or might not function correctly.

This could happen if you manually fail over one database but not the other. This scenario is less likely to occur when using automatic failover. In the event of the failure of the SQL Server service or the server or network, both databases will fail over in approximately the same time. However, what happens if one database experiences a disk failure? In that case, only that database fails over. Or, if the network is experiencing sporadic issues and the mirroring session of one database times out, but the other databases do not, one database might fail over, but not the others. Therefore, if your application consists of multiple databases, develop an alert mechanism to detect if one database fails over while others don't. In such a situation, you must manually fail over the other databases.

Based on the resource (memory and threads) utilization by database mirroring, it is a best practices recommendation to mirror not more than 10 databases in one instance. The recommended limit of 10 is only approximate and not an exact number. The limit can vary depending upon your application and workload.

Viewing Mirroring Information

SQL Server 2005 provides catalog views and dynamic management views (DMVs) to view mirroring information. It also provides System Monitor counters to view database mirroring performance information.

Mirroring state, safety level, and witness

The **sys.database_mirroring** catalog view provides database mirroring information for each mirrored database in the SQL Server instance. Join this view to **sys.databases** to get the database name. The following query displays the most commonly used mirroring metadata:

```
SELECT d.name, d.database_id, m.mirroring_role_desc,
       m.mirroring_state_desc, m.mirroring_safety_level_desc,
       m.mirroring_partner_name, m.mirroring_partner_instance,
       m.mirroring_witness_name, m.mirroring_witness_state_desc
FROM   sys.database_mirroring m JOIN sys.databases d
ON     m.database_id = d.database_id
WHERE  mirroring_state_desc IS NOT NULL
```

You can execute this query on either partner (the principal or the mirror), and you will get the same output.

The **sys.database_mirroring_witnesses** catalog view provides information on the witnesses. You can execute the following query on the witness server to list the corresponding principal and mirror server names, database name, and safety level for all the mirroring sessions for which this server is a witness.

```
SELECT principal_server_name, mirror_server_name,
       database_name, safety_level_desc
FROM   sys.database_mirroring_witnesses
```

Endpoints

Query the catalog view **sys.database_mirroring_endpoints** to find useful information regarding the endpoints, such as the status of the endpoint, encryption settings, etc.

The port number used by the endpoint is not in **sys.database_mirroring_endpoints**; it is in **sys.tcp_endpoints**. You can join **sys.database_mirroring_endpoints** and **sys.tcp_endpoints** to get the important metadata information regarding the endpoints:

```
SELECT e.name, e.protocol_desc, e.type_desc, e.role_desc, e.state_desc,

       t.port, e.is_encryption_enabled, e.encryption_algorithm_desc,

       e.connection_auth_desc

FROM   sys.database_mirroring_endpoints e JOIN sys.tcp_endpoints t

ON     e.endpoint_id = t.endpoint_id
```

Monitoring database mirroring performance

To monitor the performance of database mirroring, SQL Server provides a System Monitor performance object (**SQLServer:Database Mirroring**) on each partner (principal and mirror). The **Databases** performance object provides some important information as well, such as throughput information (**Transactions/sec** counter). Following are the important counters to watch.

On the principal:

- **Log Bytes Sent/sec**: Number of bytes of the log sent to the mirror per second.
- **Log Send Queue KB**: Total kilobytes of the log that have not yet been sent to the mirror server.
- **Transaction Delay**: Delay (in milliseconds) in waiting for commit acknowledgement from the mirror. This counters reports the total delay for all the transactions in process at that time. To determine the average delay per transaction, divide this counter by the **Transactions/sec** counter. When running asynchronous mirroring this counter will always be 0.
- **Transactions/sec**: The transaction throughput of the database. This counter is in the **Databases** performance object.
- **Log Bytes Flushed/sec**: The rate at which log records are written to the disk. This is the log generation rate of the application. It plays a very important role in determining database mirroring performance. This counter is in the **Databases** performance object.
- **Disk Write Bytes/sec**: The rate at which the disk is written to. This counter is in the **Logical Disk** performance object and represents. Monitor this counter for the data as well as the log disks.

On the mirror:

- **Redo Bytes/sec**: Number of bytes of the transaction log applied on the mirror database per second.
- **Redo Queue KB**: Total kilobytes of hardened log that remain to be applied to the mirror database to roll it forward.

- **Disk Write Bytes/sec**: The rate at which the disk is written to. This counter is in the **Logical Disk** performance object and represents. Monitor this counter for the data as well as the log disks on the mirror.

Troubleshooting

If database mirroring is not working, check the following to make sure the configuration is correct:

- The endpoints are all started. Look at the column STATE_DESC in the catalog view **sys.database_mirroring_endpoints**. The default state at the time of the creation of the endpoint is STOPPED. You can start the endpoint at the time you create it by explicitly specifying the state as STARTED. Alternatively, you can use the ALTER ENDPOINT command to start the endpoint.
- The endpoints in the principal, mirror, and witness (if any) have compatible encryption settings. Look at the column IS_ENCRYPTION_ENABLED in the catalog view **sys.database_mirroring_endpoints** for the principal, mirror, and witness endpoints. This column has a value of either 0 or 1. A value of 0 means encryption is DISABLED for the endpoint and a value of 1 means encryption is either REQUIRED or SUPPORTED. (For information on how to change the encryption setting of a database mirroring endpoint, see Endpoint encryption earlier in this paper.)
- The port numbers of the endpoints are the same as the corresponding port numbers specified in the SET PARTNER statements. The port number used by an endpoint is listed in the PORT column of the catalog view **sys.tcp_endpoints**. The port number specified in the SET PARTNER statements while establishing the database mirroring session is listed in the column MIRRORING_PARTNER_NAME of the catalog view **sys.database_mirroring**. Make sure that the port number in MIRRORING_PARTNER_NAME matches the port number of the corresponding endpoint.
- The endpoints have the correct type and role. The type and role of an endpoint are listed in the columns TYPE_DESC and ROLE_DESC respectively of the catalog view **sys.database_mirroring_endpoints**. Make sure that the type is DATABASE_MIRRORING. The role should be PARTNER if this endpoint belongs to a server that is used as the principal or mirror. The role should be WITNESS if this endpoint belongs to a server that is used as the witness.
- The user account under which the SQL Server instance is running has the necessary CONNECT permissions. If SQL Server is running under the same domain user on all the servers involved in the mirroring session (principal, mirror, witness), then there is no need to grant permissions. But if SQL Server runs under a different user on one or more of these servers, then you need to grant CONNECT permission on the endpoint of a server to the login account of the other servers.
- The remote connection through TCP/IP is enabled on all SQL Server instances involved in the database mirroring session.
- If you have a firewall between the partners and the witness, make sure the port that is used for database mirroring endpoints is opened through the firewall.

Conclusion

The performance of database mirroring is very closely associated with the type of application, transaction safety level, and network performance. Understanding the application behavior in terms of log generation rate, number of concurrent connections, and size of transactions is important in achieving the best performance. The network plays a very important role in a database mirroring environment. When used with a high-bandwidth and low-latency network, database mirroring can provide a reliable high-availability solution against planned and unplanned downtime. Across geographically distant data centers, database mirroring can provide the foundation for a disaster-recovery solution.

Appendix A Test Environment

Database servers

Two NEC Express5800/1080Xe servers (also known as NEC NX7700i / 3020M-8 in Japanese markets), each with:

- 8 processors Intel Itanium2 1.5GHz / 6-MB cache
- 8 GB RAM
- 1 Gbps network interface card

Storage

Each database server is equipped with:

- NEC Storage S2300 with two disk enclosures
- 12 physical disks in each disk enclosure
- Each disk: 36 GB @15000 rpm
- Each disk enclosure configured as one LUN with 12 disks in RAID 1+0
- One LUN for data files, one LUN for log files – for physical isolation between data and log
- One QLogic QLA2340 PCI Fiber Channel Adapter connected to each disk array

Software

- Operating system: Microsoft Windows 2003 Server™ 64-bit Edition with Service Pack 1 applied
- SQL Server: SQL Server 2005 Enterprise Edition for IA64 platform RC1 (Build 1355)
- Network Emulator for Windows for simulating network latency and bandwidth