Technical report:

Microsoft SQL Server 2000 and IBM System Storage N series with SnapMirror technology

Disaster Recovery

Document NS3108-0

July 30, 2007

Table of contents

Abstract

SnapMirror technology is efficient, low-cost, and reliable. It offers unique configuration choices and even delivers proactive data leverage capabilities. This white paper explains that IBM System Storage N series with SnapMirror establishes a user-friendly, dependable, and flexible disaster recovery solution for Microsoft SQL Server.

Introduction

This document describes techniques for setting up a disaster recovery configuration for Microsoft® SQL Server databases using IBM® System Storage® N series with SnapMirror® technology. Specifically, we cover the following issues:

- Description of the SnapMirror technology and its approach to disaster recovery
- The infrastructure required to support SnapMirror technology
- How to set up a SnapMirror for use in a Microsoft SQL Server environment
- How disaster recovery works in practice
- Resyncing the mirror following recovery from a disaster.

Hardware and software used by the test environment are:

- Two IBM N series filers and IBM System Storage N series with Data ONTAP®
- SQL Server 2000, Enterprise Edition
- Microsoft Windows® 2000 Advanced Server.

The methodology described in this paper shows how to recover a database to the same consistent state it was at when the SnapMirror shot was taken; committed transactions after the SnapMirror shot was taken are lost. If the requirement is to recover to the last committed transaction prior to the disaster, the database log file has to be located outside the filer on a medium that is accessible from the recovering system.

Description of SnapMirror technology

IBM System Storage N series with SnapMirror technology provides asynchronous mirroring of data between filer volumes. Data on the source volume is periodically replicated to the target at a user-definable time interval, with the range being from one minute to one month. At the end of each replication event, the mirror target volume becomes an exact block-for-block copy of the mirror source volume. At that point, the two volumes share identical data content and characteristics. The mirror is initialized by effectively copying the entire source volume to the target volume. Once this initial copy is complete, replication events thereafter copy only changed blocks from the source volume to the target volume. This provides a highly efficient data replication mechanism.

Architecturally SnapMirror software is a logical extension of the IBM System Storage N series with WAFL® (write anywhere file layout) file system, particularly the IBM System Storage N series with Snapshot™ feature. Using Snapshots, one can create a read-only copy of an entire filer volume. This copy is made by essentially saving only changed blocks after a particular point in time. Two sequential Snapshots can then be compared and the differences identified. Since this comparison takes place at the block level, only the changed blocks need be sent to the mirror target. By implementing the update transfers asynchronously, data latency issues inherent with remote synchronous mirroring techniques are eliminated. The elegance of these two design features becomes particularly apparent when running mirror pairs over WAN topologies.

SnapMirror scheduling is configurable by the filer administrator, and since SnapMirror creates and uses its own Snapshots, is it not possible for the database server to know when a SnapMirror file will be updated. Therefore, if the requirement is to use the database in the mirror as a reporting database or to recover from soft database error (database objects deleted or modified by operational mistake) than another snapshot has to be used. Snapshots created on the source filer are automatically migrated to mirror filer, and SQL Server can attach to the database in the mirrored snapshot.

General assumptions and requirements

In delivering this technical report, it is assumed that the reader is familiar with Microsoft SQL Server, the operation of IBM N series filers, and the Windows operating system.

The examples in this technical report assume the following:

- The name of the source filer is **bg**.
- The name of the target filer is **lg**.
- The name of the administrative user account within SQL Server is **internal** and the password of this user is **microsoft**.
- All data is stored in the **mssql** directory on **vol2** on **bg**.
- All data is mirrored to **vol2** on **lg**.

Prerequisites

Following are the prerequisites for using SnapMirror on an IBM N series filer with Microsoft SQL Server:

Filer prerequisites

- Both target and source filers must be running the same Data ONTAP version. The respective software version must be Data ONTAP (7.1 or higher).
- Both the SnapMirror and Common Internet File System (CIFS, for Windows networking) licenses must be enabled on all participating filers.
- An offline volume must be created or an existing volume must be taken offline. This volume will be the mirror volume. If using an existing volume, all premirror data on that volume will be overwritten once the baseline transfer has taken place. Because the contents of this volume will be overwritten by the SnapMirror volume data, this volume should not be the root volume.
- The source volume must not be a mirror and it must be online.
- The capacity of the mirror volume must be greater than or equal to the capacity of the source volume. The configuration of these volumes can differ geometrically (e.g., 16x18GB drives vs. 8x36GB drives), however, any geometric mismatch (where the mirror volume capacity is less than the source volume capacity) will result in a significant performance penalty.
- The source volume must be large enough to house the database(s) being mirrored.

SQL Server prerequisites

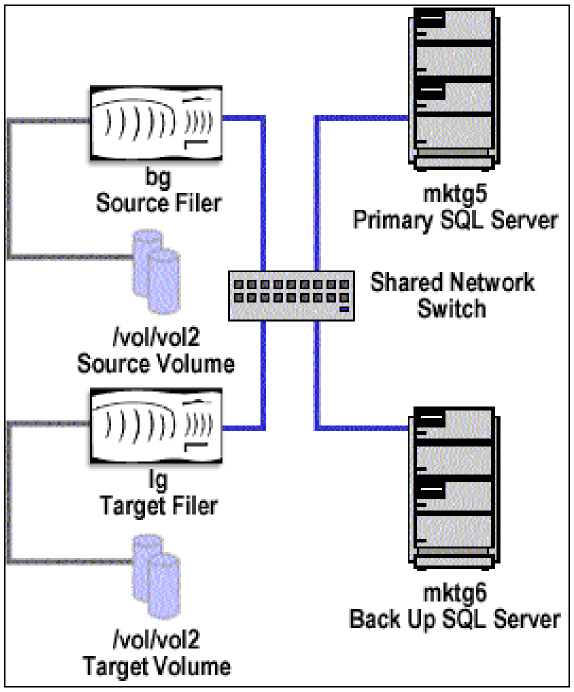
- A Microsoft SQL Server install on Windows Server is required to implement the exercises in this document.
- Both the SQL Server database data and log files must reside on the source volume.

Network prerequisites

- A network connection is required between the SQL Server machine and the filer. 100BaseT, Fiber Distributed Data Interface (FDDI), and Gigabit Ethernet are all viable options. The faster the network, the better the performance.
- A robust network connection between the source filer and the target filer is required. It must have sufficient bandwidth to accommodate the anticipated data change rate and SnapMirror software overhead.
- The network connection type should be based on the following parameters:
  - Data transmission costs between the source and target filers
  - The source volume size
  - The data change rate
  - The SnapMirror software update schedule.

In addition to the data change block transfers, each replication event requires an updated WAFL block map defined as 0.1% of the source volume size. The block map file is transferred for each mirror iteration regardless of the number of changed blocks. For example, a 250GB volume mirror will transfer a 250MB block map file plus all changed data blocks since the last Snapshot.

Following is the network connection we used to test this solution:



Setting up a SnapMirror process for SQL Server 2000 data file storage

Mirrors are as simple to set up and manage as filer volumes. Once all prerequisites have been met, the following must be done to initiate the process:

Step one (implemented on the source filer)

Add the host name of the destination filer to the **/etc/snapmirror.allow** file on the source filer. In our test environment, **lg** is the name of the destination filer, so the **/etc/snapmirror.allow** file on the source filer (**bg**) appears as follows (this file can be viewed by mapping a drive to the root volume of the filer and looking in the **/etc** directory):

```
lg
```

Note: The filer does not ship with a default **/etc/snapmirror.allow** file. Rather, it must be created using a text editor.

Step two (implemented on the destination filer)

Specify the following in the **/etc/snapmirror.conf** file on the destination filer:

- The source filer and volume
- The target filer and volume
- The maximum network bandwidth usage throttle
- A list of incremental update times in minutes, hours, days, and months.

In our testing environment, the **/etc/snapmirror.conf** file on the destination filer **lg** appears as follows (see ***Step One*** for how to view this file):

```
bg:vol2 lg:vol2 kbs=5000 * * * *
```

The above text sets up the following SnapMirror software parameters:

- The **vol2** volume on **bg** will be replicated onto the **vol2** volume on **lg**
- A 5,000KB per second maximum throttle is set on the mirror
- The SnapMirror interval is set to occur every minute of every hour of every day of every week of every month.

Note: The filer is not shipped with a default **/etc/snapmirror.conf** file. Rather, it must be created using a text editor.

Step three (implemented on both the source and destination filers)

At the command line of both the source and destination filers, enter the following command:

```
vol snapmirror on
```

It is important to note that the **vol snapmirror on** command does not persist across filer reboots. Rather, the command must also be put in the **/etc/rc** files of both source and destination filers. (Note: this command can be placed anywhere after the network interfaces are defined.) If placing the **vol snapmirror on** command in the **/etc/rc** file, following is how this file may appear (see ***Step one*** for how to view this file):

```
#Regenerated by registry Wed Apr 21 10:10:28 PDT 1999
#Auto-generated by setup Mon Apr 19 18:22:22 GMT 1999
hostname bg
ifconfig e0 `hostname`-e0 mediatype auto netmask 255.255.252.0
route add default 10.153.4.1 1
routed on
options dns.domainname 2700-1.netapp.com
options dns.enable on
...
vol snapmirror on
```

With SnapMirror turned on, the destination filer will read the /etc/snapmirror.conf file and, at the next scheduled mirror update, establish a connection with the source filer. If a baseline version of the mirror does not exist, the filer takes a Snapshot of the source volume and transfers all the data in the Snapshot from the source volume to the mirror. If a vol status command is issued at this point, the following result will appear (noting that, in this example, vol2 is our mirror volume):

Command:

```
bg>vol status
```

Result:

```
Volume   State   Status        Options
vol0     online  normal        root, nosnap=on
vol1     online  normal
vol2     online  snapmirrored
```

Subsequent updates to the mirror are made according to the schedule specified in the /etc/snapmirror.conf file.

SnapMirror software operation

SnapMirror has two distinct phases: initialization and incremental update. The initialization phase consists of a level 0 replication event in which a Snapshot is created on the source volume and then sent in its entirety to the target volume. The amount of time required to replicate the entire source volume over to the target volume depends on many factors, including the network connection. For example, a 250GB mirror initialization that would take approximately 7.5 hours to complete across a 100BaseT full-duplex link may take 1.5 hours across a gigabit link. The level 0 event serves to initialize or "seed" the mirror volume since it contains every block in the source volume as of the time the Snapshot was created.

After mirror initialization is complete, the target filer(s) examines its **/etc/snapmirror.conf** file every minute to see if there are any scheduled updates. This allows for modification of the mirror's configuration without disrupting the mirror. When an incremental update schedule time is due, a new Snapshot is taken and compared to the previous Snapshot. The different blocks and block map file are sent to the mirror target. In contrast to the level 0 initialization, the data mirrored is typically much smaller. Note that at all times the mirror target file system is in a consistent state.

Several special cases should be noted:

- If an initialization (level 0) replication event is interrupted for more than nine minutes (e.g., a network outage), it will abort. Partial level 0 events are not recoverable, so the process must be restarted.
- The KB per second maximum throttle parameter in **/etc/snapmirror.conf** can be changed at any time, and the edited values will take effect within two minutes. The exception to this is mirror initialization where a bandwidth throttle is already in effect. In this case, the bandwidth throttle cannot be modified until the initialization phase is complete or the process is interrupted.
- Incremental updates will not start until the level 0 initialization is complete.
- Any incremental updates that are missed are simply skipped. Subsequent incremental updates transmit any skipped data, so no data is lost.
- Incremental updates in progress will run to completion according to the configuration settings and available network bandwidth. If a new incremental update is scheduled to start while an existing update is in progress, it is considered a schedule miss and this is skipped.
- The SnapMirror process can be stopped and restarted at any time by issuing the following command sequence on either filer:

  ```
  vol snapmirror off
  [arbitrary time interval]
  vol snapmirror on
  ```

  As long as the target volume remains read-only during the time between turning the SnapMirror process off and on, the mirror remains intact.

- To break the mirror, SnapMirror must be turned off, at which time the target volume will be placed into read/write mode. In order to turn SnapMirror off, the following command should be executed on the destination filer (in our testing environment, **vol_name** is **vol2**):

  ```
  vol options vol_name snapmirrored off
  ```

  Note: The SnapMirror file can only be broken if a SnapMirror event is not in process.

Disaster recovery—breaking the mirror

After creating a SnapMirror relationship between two filers we implemented the following test:

1. We created the following table on the database **test_db**. The database's data and log files were located in the **mssql** directory on **vol2** on **bg**:

   ```
   DROP TABLE hammer_tab;

   CREATE TABLE hammer_tab
       ( value INT NOT NULL );
   ```

2. We then created the following stored procedure in the table hammer_tab:

   ```
   USE test_db
    GO;
   CREATE PROCEDURE add_data
   AS
   DECLARE @counter INT
   SELECT @counter = 1
   WHILE ( @counter < 100000000000 )
   BEGIN
     INSERT INTO hammer_tab
     ( value )
     VALUES
     ( @counter );
   PRINT 'Value is: ' + convert ( varchar(6), @counter1 ) SELECT @counter =
   @counter + 1
   END
   ```

3. We then executed **add_data** from the SQL Server's Query Analyzer. With the procedure running, we allowed several SnapMirror events to occur.

4. Toward the end of the ten-minute procedure run, we changed the SnapMirror interval in the **/etc/snapmirror.conf** file to a less frequent mirror. We did this because we were about ready to simulate a filer/SQL Server failure followed by a recovery of both servers. We did not want a SnapMirror event to initiate once the filer recovery had completed. If an event was triggered, we would not be able to break the mirror.

5. Next we simulated the above-mentioned disaster with a goal of mimicking a complete loss of service of both the filer and the database server. We did this in the following manner:

   - Failed the source filer (**bg**) by powering it down
   - Failed the SQL Server by turning off the MSSQLServer service while the query was still running.

   The failure of both the filer and the SQL Server resulted in the following activity from within the Query Analyzer (note that the last row inserted has a key value of 150942):

   ```
   (1 row(s) affected)
   Value is: 1
   (1 row(s) affected)
   Value is: 2
   (1 row(s) affected)
   Value is: 3
   (1 row(s) affected)
   Value is: 4
   ... Many more like this ...
   (1 row(s) affected)
   Value is: 150939
   (1 row(s) affected)
   Value is: 150940
   (1 row(s) affected)
   Value is: 150941
   (1 row(s) affected)
   Value is: 150942

   [Microsoft] [ODBC SQL Server Driver] [Named Pipes] ConnectionRead
   (GetOverLappedResult()).
   [Microsoft] [ODBC SQL Server Driver] [Named Pipes] Connection broken.

   Connection Broken
   ```

6. After changing the mirror interval, we broke the mirror on **vol2** (making lg's **vol2** read/write) by issuing the following command on **lg**:

   ```
   vol options vol2 snapmirrored off
   ```

7. We then created a CIFS share (**sqldb**) pointing to the directory (**mssql**) on the SnapMirror volume (**vol2** on **bg**) in which the data resides.

8. After creating the share, we created a new SQL Server database (**test_db_new**) that pointed to the data and log files that were on **lg**'s now read/write volume **vol2**. We did this by executing the following statement from an ISQL prompt:

```
dbcc traceon ( 1807 )
go
sp_attach_db 'test_db_new', '\\lg\sqldb\test_db_data.mdf',
'\\bg\sqldb\test_db_log.ldf'
go
dbcc traceoff ( 1807 )
```

9. We then queried the **hammer_tab** table in both **test_db** and **test_db_new** to see if the data in the table on each database matched. Following are the query scripts and their respective result sets:

   Query on test_db:

   ```
   USE test_db
   SELECT * FROM hammer_tab
   ```

   Result set from query on test_db:

   ```
   value  time
   1
   2
   3
   4
   ```
   ... Many more rows incremented by one ...

   ```
   150939
   150940
   150941
   150942
   (150942 row(s) affected)
   ```

   Query on test_db_new:

   ```
   USE test_db_new
   SELECT * FROM hammer_tab
   ```

   Result set from query on test_db_new:

   ```
   value  time
   1
   2
   3
   4
   ```
   ... Many more rows incremented by one ...

   ```
   130941
   130942
   130943
   130944
   (130944 row(s) affected)
   ```

Note that the key value for the source database matches precisely to that which was reported to the client. This is consistently our experience: When an IBM N series filer experiences a dirty shutdown, no transactions are lost, although the SQL Server may apply a recovery to the database on the source filer (**bg**). However, the target database did lose approximately 20,000 rows over the ten minutes that the test was run. This is to be expected. The mirror interval was set to 60 seconds, and the mirror relies on consistency points. Thus, the mirror can be as much as two minutes out-of-date with the interval set in this way. (A longer interval would result in more lost transactions, but also lower overhead.)

In our example, a total 150,942 rows were processed on the source database table. This is a rate of 15,094.2 rows per minute. Given that the mirror can be out-of-date by minutes, the number of rows that did not mirror over to the target database table could have been as high as 30,188.4 (20% of the table). The bottom line is that the target database can be created as a mirror of the source database, with a loss of only a few very recent transactions. For a disaster recovery solution, this is perfectly acceptable to the vast majority of customers.

Resyncing the mirror

If a volume is migrated, to reconfigure a mirror, simply change snapmirror.conf to match the source system/volume/qtree name and configure the new source to allow transfers to the destination (use the snapmirror.allow or /etc/snapmirror.allow option). That's it. There's no need to break and/or resynchronize the mirrors. SnapMirror picks up where it left off.

While the ability to restart a broken mirror may be in future release, presently, the method for reestablishing a mirror is to reinitialize the mirror. If the target volume has been brought online and then written to, this may need to be done twice.

- Once to replicate the data back to the source volume. (In this case the mirror relationship between the two filers is flipped.)
- Once to reinitialize with the mirror relationship back to normal.

An alternative is to back up the target volume's data to tape, and then restore that data onto the source filer. This avoids the need to reinitialize the mirror twice; however, one cannot "seed" a mirror from tape.

Conclusions

The IBM System Storage N series with SnapMirror technology provides compelling advantages for the SQL Server DBA seeking to support disaster recovery of a mission-critical SQL Server database. Specifically:

- The network traffic generated by the mirror process can be throttled to allow support for WAN connections.
- The mirror interval is user-configurable and can be changed on-the-fly.
- In the event of a disaster on the source side of the mirror, the target side can be easily brought online, and the SQL Server database can return to normal use after a brief recovery process.
- The near real-time nature of SnapMirror software means that the source filer will still operate normally when the network link to the target filer is broken. This is not true of synchronous mirroring, where the mirror link is essential to continued operation of the source machine.

Using SnapMirror technology, the DBA can assure the customer that the only transactions that will be lost are those that occur between the mirror events.

Support

Microsoft created dbcc trace flag 1807 to enable the storage of SQL Server databases on IBM N series filers. This flag is required to successfully migrate or create SQL Server 7.0 or SQL Server 2000 databases on a filer. Databases can be placed on any network-attached device using this flag.

Caveats

Though this document provides a comprehensive overview of SnapMirror technology, additional reading may be beneficial. See the IBM System Storage N series ***Systems Administrator's Guide*** or the ***Data Protection Online Backup and Recovery Guide.***

Trademarks and special notices