Best Practices: Using TMF to Implement Business Continuity/Disaster Recovery

Richard Carr
Carr Scott Software Inc.

Introduction

One of the common requirements of a Business Continuity/Disaster Recovery (BC/DR) strategy on NonStop systems is to quickly provide the production database on a backup system. To implement that strategy, one must chose and deploy a database replication solution, which may or may not be based on TMF auditing. This article shows how that choice may affect overall system performance.

Disaster Recovery Options

Although some customers may be able to implement a BC/DR strategy using BACKUP and RESTORE, those with the highest availability requirements will seek a method that maintains a near-real-time mirror of the database on a remote backup system. Thus, there are at least five companies, including HP, that provide products to replicate a NonStop database for BC/DR.

When choosing such a product, one of the fundamental options is whether to base replication on capturing database updates by interception or by TMF auditing the database.

Interception

Interception requires attaching a user library to every program that might update the database. The library receives all calls to database update procedures, such as WRITE(X), WRITEUPDATE(X), CONTROL, etc., and then performs two I/O operations for each database update: one to update the actual database and one to write a description of the update to a transaction log.

The advantage of interception is that it works for an unaudited database. Base24, the most widespread application on the NonStop platform, uses an unaudited database. BC/DR is a high priority for many of those customers.

The primary disadvantages of interception are (1) the extra I/O operations required in each application process, and (2) the possibility of error, in ensuring every program is always configured with the interception library. If a new version of a program does not have the interception library, it will update the primary database but not the backup database, and you may never know that happened.

TMF Auditing

HP Nonstop™ Transaction Monitoring Facility (TMF) has been a fundamental part of the NonStop since 1980, so the reader is assumed to have basic knowledge of this critical subsystem. The key features of TMF that are important to this discussion are:

1. It is impossible to update an audited file unless TMF writes an audit trail record describing that update.
2. TMF auditing is implemented in the disk process and provides an extremely efficient method to capture all database updates.
3. Because all database updates are captured in the audit trail, the disk process can defer updating the actual data files without loss of data even in case of a system failure. This is the most reliable way to eliminate physical I/O operations without risk.

Other advantages of using TMF auditing include

1. Reliable **online** backup and recovery of audited data, using standard TMF roll forward recovery of a damaged file or volume. Online backups of unaudited files that are being updated are worse than useless because you have no idea if they can be used in event of a failure.
2. Transaction backout keeps a database consistent if applications are properly programmed to invoke TMF transactions at business transaction boundaries.

The primary disadvantage of TMF auditing is the requirement that all programs that update the database must be programmed to make calls to BEGINTRANSACTION and ENDTRANSACTION. For complex applications, especially those designed without TMF in mind, adding transaction calls can be difficult to do. But, as you might expect, there is a solution to this problem: AutoTMF.

AutoTMF

HP NonStop™ AutoTMF automatically provides TMF transactions when required, but without requiring any change to the application code or logic. It is currently in use in hundreds of customer applications, many of which are running Base24. Some of these customers adopted AutoTMF to support TMF-based replication but many of them are using AutoTMF solely to improve performance.

AutoTMF uses an interception library, but with an important difference: if you forget to attach AutoTMF to a program, that program will fail when it tries to update an audited database. Any mistakes will be quickly found and corrected without losing updates to the backup database.

In terms of performance, online backup and recovery, and capturing database updates, AutoTMF works just as effectively and efficiently as explicit use of TMF. But, AutoTMF does not know the boundaries of a business transaction, so it never uses transaction backout. Thus, it will behave the same as the original application program.

Finally, AutoTMF is ideal for converting a large collection of non-TMF-aware programs to use transactions because it allows incremental reprogramming of the key database access programs without requiring all to be changed simultaneously.

TMF Performance

A common reason for not adopting TMF is the fear of taking a performance hit. TMF must manage transactions generated on many cpus, implement two-phase commit, and send all the audit from many disk processes to the audit trail disk. It would

seem impossible to do all that work and still have acceptable performance.

But, it is well known that using TMF almost always improves performance. There are four primary reasons for this:

- TMF overhead is minimal. Implemented as part of the NonStop OS kernel and disk process, transaction processing requires a low level of resources.
- Audit records sent from the database disk process to the audit trail disk process are blocked together, using a technique called boxcarring. A few messages can support a large number of transactions.
- Audit trail writes are also boxcarred. Audit for many transactions from many disks is collected and written to the end of the audit trail with a single I/O.
- Most importantly, the database disk processes can eliminate physical I/O operations if updates are audited. It can do this with no possibility of data loss. If a system fails, "lost" updates are reapplied from the audit trail.

TMF performs best if it has many parallel processes generating transactions. For a batch program, however, it is best to perform many updates in each transaction.

Since there are still skeptics about TMF performance, the performance study in the next section attempts to quantify these claims.

Performance Study

The sole purpose of this study is to demonstrate the improved performance of a TMF-audited application when performing typical business operations.

This study emulates a system performing a fundamental operation of a POS/ATM application: the credit authorization. In the online environment, it is one of the most common operations. A real application would perform other types of operations to deal with exceptional cases, but the I/O of these operations is essentially similar.

The study is not intended to predict the transaction rates of any actual application. Our goal was only to perform the same types of database I/Os that predominate in an actual application, and measure the relative improvement of using TMF auditing.

The Test Environment

The test system was an NB50000c (NSE-M) system with four dual-core Itanium cpus and sufficient disk storage and memory. The operating system version was J06.15.00. TMF was configured with a single master audit trail.

Application Emulation

A simple transaction generator program emulates the POS authorization by performing I/O to a representative database. Each process attempts to create a specific workload and increasing numbers of the processes are run to create an increasing workload.

The representative database consisted of two files:

(1) Terminal Definition File (TDF) – a Relative file with two alternate keys. The file had four partitions. Record length is 4072 bytes. The test file contained 10,000 devices. The TDF is accessed via an alternate key.

(2) Transaction Log File (TLF) – an Entry-sequenced file with four alternate keys. Record length is 998 bytes. Each alternate key file had two partitions.

Care was taken to provide for parallelism when accessing the database. Each file partition was placed on a separate physical disk and disk activity was evenly distributed across all partitions. Very large disk caches were configured.

The emulation program performed a sequence of database operations to authorize a credit card. There is a fixed sequence of operations to perform each authorization. These operations include:

(1)Receiving an authorization request from a "terminal".

(2)Reading (via an alternate key) and locking the terminal record in the TDF.

(3)Sending a message to an authorization server. This does not require a database I/O.

(4)Updating the TDF record.

(5)Writing a record to the transaction log TLF.

(6)Replying to the requester.

This authorization transaction was the most common sequence of I/O operations, as determined by tracing an actual production application.

Testing Protocol

Each server process is configured to perform a fixed number of transactions per second. Increasing numbers of server processes were run until the system was saturated and a maximum transaction rate was achieved. The "load" is the number of transactions that the application attempts to process.

In each test run, the database and servers were configured to use one of the following setups:

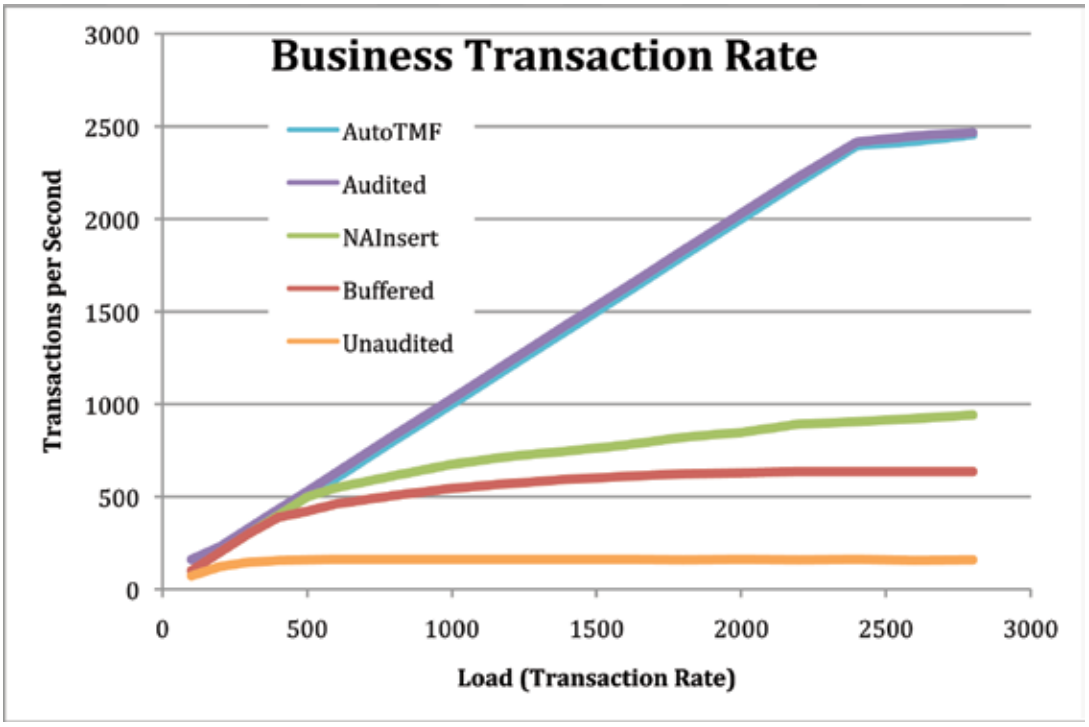
- **UNAUDITED** – No files were either audited or buffered. No TMF transactions were created.
- **BUFFERED** – Files were buffered. No TMF transactions were created.
- **NAINSERT** – Files were buffered and enabled to use the NONAUDITEDINSERT option . No TMF transactions were created.
- **AUDITED** – Files were audited. Program performed one TMF transaction for each business transaction.
- **AUTOTMF** – Files were audited. Program did not perform TMF transactions. AutoTMF created one automatic TMF transaction for each business transaction.

Each program reported its transaction rate and average response time. Response time is the wall-clock time from after receiving the authorization request to after sending the reply. The results from each program was collected and collated to produce the total transaction rate and overall mean response time.

Test Results

The following charts show the results of 85 separate test runs, consisting of 17 different loads and the 5 different database/application configurations.

The first chart shows the growth of the completed transaction rate as the load is increased. The transaction rate of an unaudited database is severely limited by the physical I/O required for

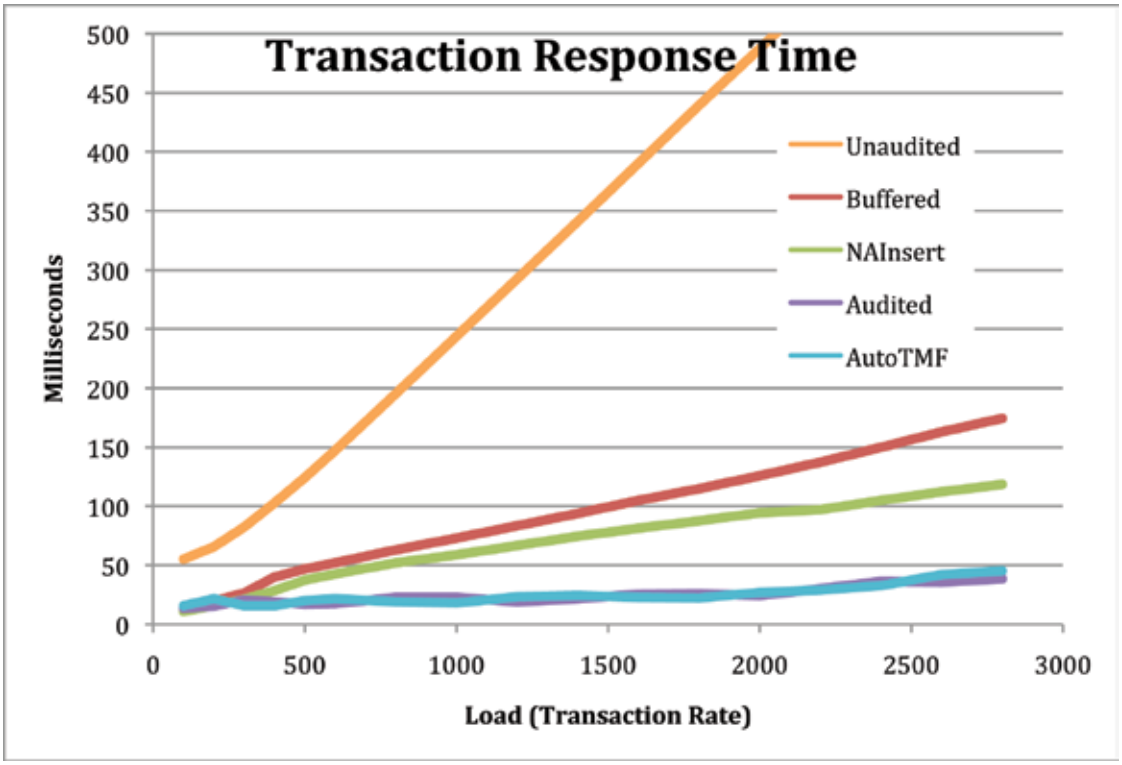


each transaction. Although the database was partitioned, the maximum transaction rate was about 150 TPS for unaudited access.

Buffering the database greatly increases the transaction rate to about 630 TPS, but at some small risk of massive data loss in case of a system failure. NAINSERT further improves performance to 950 TPS. NAINSERT streamlines inserts to the TLF, but doesn't significantly improve inserts to the alternate key files.

Maximum performance (2450 TPS) and maximum data integrity is achieved with the use of TMF transactions and an audited database. With auxiliary audit trails, even higher levels can be achieved.

The second chart shows the effect that a high transaction rate will have on the end user of the system. As more and more load is placed on the system, users will have to wait longer and longer. This is particularly dramatic for the unaudited database.



Configuring BUFFERED and NAINSERT dramatically decrease the degradation of response time with increasing load, but the use of audited files provides the fastest response time.

Much time was spent studying the BUFFERED and NAINSERT measurements in an attempt to eliminate any bottlenecks and improve performance. But, the fact remains that, even for buffered files with large caches, DP2 must perform physical I/O to maintain a consistent file structure. No one wants to have the "broken" files that characterized the days of DP1.

NAINSERT showed a marked improvement over BUFFERED, by reducing the cost of inserts to the Entry-sequenced TLF. But, the bottleneck simply moved to the inserts to alternate key files. If one eliminated all the alternate key files on the TLF, the NAINSERT results would have been slightly better than AUDITED; both can achieve more than 3000 TPS with response times less than 40 ms. But, a survey of some major Base24 installations indicated that the alternate key files are necessary for application functionality.

Benchmark Considerations

In benchmarks and other performance studies it is easy to overlook extraneous factors (such as page-faulting or small cache sizes) that lead to false conclusions. A diligent attempt has been made to model and measure the authorization transaction without such factors becoming significant.

Measure performance data for the test runs (available from the author) was analyzed for problems such as excessive CPU busy, page faulting, lock contention and excessive disk I/O. No such problems were found.

With a lot of effort to improve the BUFFERED and NAINSERT results (through partitioning and adjusting cache sizes), the maximum transaction rate was increased from about 550 TPS to 950 TPS. The AUDITED rate of 2400 TPS was achieved on the first run with no tuning whatsoever.

Note that this study did not include any use of interception to collect updates for replication of unaudited data, so the impact on performance would need to be included in any consideration for BC/DR use. With audited files, the updates are already being collected by TMF and can be readily used for reliable database replication.

Conclusion

For real-time replication of a database to a remote backup system, having a TMF audited database not only offers significant operational, reliability, and integrity advantages, it also improves overall system performance.

[i] NONAUDITEDINSERT is a special DP2 setting that improves performance when writing to an entry-sequenced file that is also buffered. Like buffered, updates are cached in memory and may be lost if a system failure occurs.

Richard Carr is a co-founder and partner of Carr Scott Software Inc and designed and implemented Escort SQL, HP Nonstop AutoTMF, and HP NonStop SDR.

Prior to 1995, he was a Technical Director for Tandem Computers where he worked for 15 years in Guardian kernel development, the High Performance Research Center, and Tandem Labs. He designed and/or implemented many products, including the NSK memory manager, parallel processor load, Global Update Protocol, expedited message system, DSAP/DCOM, Subsystem Programmable Interface (SPI), and performance features of Remote Database Facility (RDF).

His degrees include a Ph.D. in Computer Science from Stanford University (1981) and a B.A. in Mathematics from Georgetown University (1968). He holds four patents for innovations in NonStop software products, including the first software-only patent ever awarded for NonStop.