the Availability Digest

# Virtual Transactions with NonStop AutoTMF

April 2007

In previous articles, we have discussed various ways to keep database copies in an active/active application network synchronized. A frequently used method to accomplish this is data replication.[1] Data replication engines require a log of database changes that they can use to replicate changes from a source database to a target database. Transaction monitors generate transaction logs that are ideal for this purpose.

Transactional applications benefit from many other advantages as well, including guaranteed database consistency, higher performance, and the capability to recover lost or corrupted data.[2]

However, many older applications were written as nontransactional applications and cannot share in these benefits. NonStop AutoTMF bridges this gap for applications written for HP NonStop servers. It seamlessly converts nontransactional applications to transactional applications.

## What is a Transaction?

Simply stated, a transaction is a group of operations that are so closely related that either all must be performed or none must be performed. In data processing applications, we concern ourselves with database and file operations since other functions such as printing or sending messages cannot be revoked once they have been executed.

In an application program, the operations comprising a transaction are bounded by a begin statement of some sort and an end statement. If the end statement is successfully reached, the transaction is *committed*. That is, it is made permanently to disk and will not be lost even should the system fail (short of a physical destruction of the storage device). If the end statement is not reached due to some sort of error, the transaction is *aborted*; and the effects of all previous operations within the transaction are reversed.[3]

## Why Transactions?

Transactions ensure that the database is always consistent. This is because no partial set of related operations are ever executed.

The database changes within the scope of a transaction do not really need to be physically written to disk to ensure that they are durable. It is only necessary to record enough information on disk so that the changes can be reconstructed, if necessary. This significantly improves application performance.

---

[1] Asynchronous Replication Engines, *Availability Digest*; November, 2006.
Synchronous Replication, *Availability Digest*; December, 2006.

[2] See our article in this issue entitled Transaction-Oriented Computing.

[3] See the book review, Transaction Processing: Concepts and Techniques, in this issue of the *Availability Digest.*

In many transaction management systems, such as HP's NonStop TMF, change recording is accomplished by maintaining a log of all changes (the *audit trail* in NonStop's case). This transaction log typically contains before and after images of each record whose changes have been committed. It improves application performance because only the serial log has to be written, not the individual random data changes which can be applied later.

The transaction log can be used to recover lost or corrupted transactions and ensures the durability of the database.

Furthermore, the transaction log becomes the source of changes to be replicated to other systems for disaster tolerance or active/active purposes.

## Nontransactional Applications

If transaction-oriented computing provides so many benefits, why are there applications currently in operation that are nontransactional? There are many reasons:

- Some applications were developed before the advent of transaction management facilities.
- Some applications were developed after this time but before significant performance problems were solved.[4]
- Some later applications were developed without transaction protection because of a misunderstanding of transaction processing technology and benefits.

Suffice it to say, there is a large body of nontransactional applications, currently deployed in the field, that could benefit if they could be made transactional.

## NonStop AutoTMF

Moving nontransactional applications to transactional is the role of NonStop AutoTMF. AutoTMF was developed by Carr Scott Software and is marketed and supported worldwide by HP as NonStop AutoTMF.

AutoTMF watches all disk activity for selected programs so that it can automatically generate transactions that group related disk activity. When AutoTMF detects the beginning of a business transaction, it will insert a *begin transaction* command in the I/O stream being sent to the operating system. Later, when it detects that the transaction is complete, it will insert an *end transaction* command to commit the transaction.

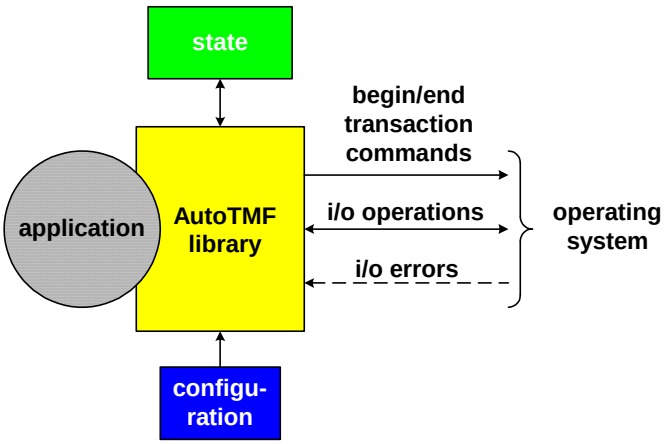
It will create transactions only for database operations on audited files. It ignores operations on nonaudited files, passing these through directly to the operating system for execution.

Some applications are not candidates for AutoTMF. For instance, applications that bypass normal operating system functions by running in privileged mode cannot be serviced by AutoTMF.

[4] In an early Stratus manual, users were warned about using transactions because of performance problems unless the reasons to do so were compelling.

### *Intercept Library*

AutoTMF is implemented as an intercept library that is bound to the application program at compile time. It comes in the form as a user library for non-Integrity systems and as a DLL for Integrity systems. It makes no changes to the program source code and is therefore totally non-invasive to the program.




The AutoTMF library intercepts all I/O calls to the operating system so that it can decide on transaction boundaries. It passes on any I/O operations that are not pertinent, such as operations on nonaudited files and communication calls.

It monitors the other I/O calls to maintain a transaction state for the application. This state contains such information as the number of open files and the number of locked records. It uses this state to help it determine when to generate a begin transaction or end transaction command. Once a transaction is started, all disk I/O to audited files is included within the scope of that transaction.

AutoTMF will never abort one of its automatic transactions as this would undo an I/O operation that the application intended to execute. In addition, it intercepts all operations, including traps and other exceptions that could terminate the process to ensure that all transactions are committed.

### *Configuration*

AutoTMF has numerous configuration parameters that can be set to control its behavior. It can be configured to create automatic transactions for all files and programs or for just specified files or programs. For instance, log files are usually not audited because if the application abended or the system crashed any uncommitted log records would be backed out and lost. However, one can configure AutoTMF to use a separate transaction for log files and to immediately commit log writes.

The maximum number of updates prior to a commit can be specified as well as the maximum amount of time since the last begin transaction command. Should either of these limits be reached, AutoTMF will automatically commit the transaction. These parameters are especially useful to add transaction capability to batch processing programs.

The level of isolation can be set. The impact of this parameter is described later under Options. The level of isolation is used to refine events that will generate a commit of the transaction.

### *Starting a Transaction*

If there is no current transaction in progress, AutoTMF will start a new transaction on any one of a number of events that signals that database activity is about to begin. For instance, these include:

- A lock for a file or a record is requested (such as a read with lock).
- A write (insert), update, or delete command is executed.

### *Committing a Transaction*

The logic that evaluates I/O events to determine when an open automatic transaction is to be committed is the heart of AutoTMF.[5] This logic is driven by the transaction state table as each new I/O operation is analyzed. Events and states that will cause a transaction commit to be generated include:

- An unlock operation that releases the last lock being held is executed.
- The last open file is closed.
- The maximum update count has been reached.
- The maximum time since the begin command has been reached.
- The process replies to a message (presumably a reply to the request that was just processed).
- A read on $RECEIVE is issued (presumably looking for the next request).
- The process terminates.

### *AutoTMF extended to OSS applications*

AutoTMF also now supports automatic TMF transaction protection for Open System Services (OSS) programs developed in C, COBOL, and Java™ classes that use the JEnscribe EnscribeFile class on HP Integrity NonStop servers.

## Options

AutoTMF has several optional modes that can be invoked.

### *Separate Transactions*

Normally, once a transaction has started, all changes to all audited files and tables are included in the scope of that transaction.

However, AutoTMF can be instructed to create a separate transaction for each audited file or table as a change is received for it. This is useful in some cases to eliminate lock contention in programs that have complex lock behavior. Each transaction can be committed as soon as there are no outstanding locks on the file or table involved in the transaction.

### *Isolation*

AutoTMF supports three levels of transaction isolation for a process. These isolation levels are intended to reduce the effect of AutoTMF on other processes with which the process is interacting.

> *Weak* – This is the default level of isolation and generates commits as described above.
>
> *Normal* – A commit is generated if the process sends a message to another process. This takes care of the case in which a requesting process, for instance, sends an intermediate transaction to another server process. The new server may attempt to lock a record already locked by the sending process, resulting in a deadlock. By committing the transaction before it is sent, deadlocks of this nature are avoided.

[5] Carr, R. W., Schilling, B. E., Corbeil, J. C., Scott, H. P., Automatic Transaction Management, United States Patent Application 20050102250; May 12, 2005.

*Strong* – This isolation level is the same as normal isolation with the addition of a commit being generated if a change is to be made to an unaudited file. This is required if there is a chance that the unaudited update might be later undone by the application.

***Audited File Creation***

The configuration file can specify the names of files that, if created by a program, are to be created as audited files.

***Audit Attribute Concealment***

In some cases, a program will ask for the attributes of a file. It is not expecting that one of the attributes will be "audited" since it thinks that all of its files are unaudited.

AutoTMF will hide this attribute from the program when it returns the attributes given to it by the operating system.

***Bulk Transfers***

Especially for batch programs, it is possible that the program will be writing blocks too large for the transaction manager to handle. This option instructs AutoTMF to break large blocks into smaller blocks that can be handled.

***No-wait Commits***

With no-wait commits, AutoTMF will return a commit complete indication to the program as soon as the program issues the ENDTRANSACTION command without waiting for the transaction to actually commit. This will improve the performance of the program. However, the next transaction will not be started until the previous commit has completed.

## I/O errors

Any I/O operation can generate an error. If the error is for an I/O operation generated by the program, that error is simply returned to the program. However, if the error occurred on a begin or end transaction command issued by AutoTMF, AutoTMF will close the current automatic transaction, if any, and will terminate the process.

## Experience with AutoTMF

Though not confirmed by formal testing, Carr Scott estimates from field experience that AutoTMF generates between one and two transactions for each real business transaction.

Many customers report significantly improved performance of their applications. After installing AutoTMF, Barclay's Bank measured a throughput capability of 650 transactions per second on a pair on NonStop 72014s. The peak load which they had been handling was 230 transactions per second.[6]

Some very large and popular third-party applications are for the most part nontransactional. A good example is Base24 from ACI, used by financial institutions around the world. AutoTMF has been successfully used to add transaction capability to applications such as Base24.[7] As a result, performance is increased, and the applications database can be replicated to a remote site for

[6] Business Continuity Customer Experiences, 2002 ETUG panel presentation, www.itug.org.

[7] P. J. Nye, Using AutoTMF, TMF, and RDF to Enable Disaster Recovery for Base24 Systems, phil@cardlink.co.uk; undated.

disaster tolerance. Barclaycard Merchant Services is one such organization that has gone this route.[8]

## NonStop AutoSYNC

Another requirement for managing an active/active application or a migration to a new system is to be able to replicate nonaudited files. Carr Scott Software is also the developer of HP's NonStop AutoSYNC.

AutoSYNC will monitor a list of files supplied to it and will replicate them to a remote system should they be changed. This is intended for files that are seldom updated, such as edit and object files or static database files or tables that see little update activity since the source file needs to be quiesced during the replication and since the target file is not available until the replication is complete.

With AutoSYNC and an appropriate replication engine for maintaining synchronization of active databases, all of the systems in an application network can be maintained in the same state.

## Carr Scott Software

Founded by Dr. Richard Carr and Harry Scott in 1995, Carr Scott (www.carrscott.com) is a privately held company which focuses on enabling software for HP NonStop servers. Located in Duxbury, Massachusetts, and Cupertino, California, Carr Scott maintains strong technology sharing agreements with HP. Their AutoTMF and AutoSYNC products are jointly marketed by HP.

Other products of Carr Scott include Escort SQL, SQL database middleware for Enscribe applications which allows NonStop Enscribe files to be transparently replaced with NonStop SQL tables. Escort Ranger analyzes Enscribe files and NonStop SQL tables to determine the optimal file partitioning ranges. Escort Journaling allows online database loading during an Enscribe to SQL migration by capturing changes that occur during the online database load so that they can be applied to the target file at the completion of the load.

Carr Scott products are used by over 300 NonStop customers worldwide.

[8] http://www.carrscott.com/barclays.pdf.