Transaction Tracking Services

NetWare Transaction Tracking Services (TTS) is a feature that ensures data integrity on files that otherwise would be corrupted when updates on the files are interrupted by such things as hardware failures or power outages. Transaction is defined as a set of one or more writes that must be completed together to maintain file and database integrity. TTS guarantees that all writes within a transaction will be completed or none will be completed. This chapter is divided into the following sections:

- Introduction
- Transaction Tracking Types
- Record Locking
- Transaction Backouts
- Applications, the User, and TTS

Introduction

The NetWare Transaction Tracking Services allow multiple stations to have the protection of transactions while concurrently updating database files. If a workstation fails during a transaction, the database changes made by that workstation can be cancelled or backed out, without affecting other workstations.

Transaction tracking is most useful in multiuser software that employs record locking, such as database applications. It is less useful in single-user applications that deal primarily with entire files instead of records, such as word processors or spreadsheets.

The Transaction Process

The following steps describe how TTS tracks each write within a transaction:

1) An application writes new data to a file on a file server.

2) The file server stores the new data in cache memory. The target file on the file server hard disk is unchanged.

3) The file server scans the target file on the file server hard disk, finds the data to be changed (old data), and copies the old data to cache memory. The file server also records the name and directory path of the target file and the location and length of the old data (record) within the file. The target file on the file server hard disk remains unchanged.

4) The file server writes the old data in cache memory to a transaction work file on the file server hard disk. This transaction work file resides at the root level of the SYS volume on the file server. The file is flagged System and Hidden. The target file on the file server hard disk is still unchanged.

5) The file server writes the new data in cache memory to the target file on the file server hard disk. The target file is now changed.

The file server repeats these steps for each write within a transaction, and the transaction work file grows to accommodate the old data for each write. If the transaction is interrupted, the file server writes the contents of the transaction work file to the target file, thereby restoring the file to pretransaction condition. In effect, the file server backs out the transaction.

Installation and Operation of TTS

This section gives some guidelines for TTS installation and operation to ensure that your system runs smoothly.

Work Files and Work Volume

During NetWare installation, you must specify which volume on the file server will be used for transaction work files. This work volume must have plenty of disk space and, if possible, be located on a different disk channel than the database files. If it is located on a different disk channel, backout data can be written simultaneously with database updates.

Transactions per DOS Task

A file server can monitor 100 to 10,000 transactions at a time. This value is configurable with SET. (The default is the maximum.) However, a file server can monitor only one transaction at a time for each workstation DOS task. If a task sends several transactions to a file server rapidly, the file server queues the transactions and services them one at a time.

Transaction Length

Transactions can be any length and contain any number of write requests. However, it is important to remember that large transactions require more space for the transaction work files. For example, a transaction in which an application truncated a 10M file to 1M, the transaction command would require over 9M of transaction work file space. It takes the file server about twice as long to process a disk write request to a transaction file as it does to process a conventional write request.

Just as exclusive locks should not be in force for long, transactions should not be active for long. Although transaction work files are reused, they cannot be reused until all transactions to which they correspond are completed. And if there are many large transactions or transactions of long duration, the transaction work files must be extended, consuming more space on the volume. It is unlikely that the transaction work volume would run out of space, but it is possible. If the work volume runs out of space, transaction tracking is automatically disabled and the following message appears at the file server console:

ªTransaction Tracking disabled because the volume is full.º

This message is issued when the volume containing the transaction work files becomes full. Processing will continue as usual; however, transaction tracking is disabled from then on (the same as if disabled from the file server console).

Any transactions that write after transactions are disabled cannot be backed out if the station or file server fails. After freeing more space on the transaction work volume, transactions may be turned on again using the file server console command ENABLE TRANSACTIONS.

Disk Cache Buffers

Transaction tracking uses extra disk cache buffers. Ideally, you should have at least 8K of disk cache buffers per station that uses the file server. An insufficient number of cache buffers causes thrashing.

Marking Files Transactional

Transaction Tracking occurs only on the files that have their transaction flag bit set. You can modify the bit with the NetWare FLAG utility, or applications can modify it using the File System APIs (NWSetFileAttributes). When this bit is set, files are referred to as transaction files. Work files or report files probably should not be marked transactional.

A file marked transactional cannot be deleted or renamed. To delete or rename such a file, flag it non-transactional. Usually, a database file is deleted or renamed before being compressed or used to generate a newer database. Forcing the user to flag a transaction file non-transactional makes the user more aware of transaction tracking. It also reminds the user to flag the new database file transactional. The application should handle this situation if it is TTS aware.

File Server System Files

The file server uses transaction tracking when updating the system files. This includes bindery and queue management files. Though no users are running TTS applications, the file server may require some transaction backouts to preserve system file integrity if the server goes down abnormally.

Spanning Multiple File Servers

Transactions can span multiple file servers. However, the explicit transaction calls must be issued to each server individually. Use the serverConnID parameter to specify which server gets the explicit transaction requests.

When spanning multiple servers, you must also consider backout windows. One file server may crash before committing the transaction, while another does not. A workstation can send an NWTTSEndTransaction to one file server, but crash before sending it to another. The application program must handle these problems.

Novell recommends that implicit transactions not span multiple file servers.

Non-Transactional Servers and TTS

A file server that does not support transaction tracking still returns successful completion codes when explicit transaction calls are made. This means that applications can be modified to use explicit transaction calls and still function if the application uses a NetWare server that is not supporting transaction tracking.

Transaction Tracking Types

There are two categories or types of transaction tracking: explicit and implicit. Explicit transaction tracking begins when an application makes an
NWTTSBegin Transaction function call and ends when an application makes an NWTTSEndTransaction function call or an NWTTSAbortTransaction call.

Implicit transactions begin automatically when an application does its first logical or physical record lock on a transaction file. (By default, an implicit transaction begins on the first lock.) When all records are unlocked, the file server assumes that the transaction has ended.

The NetWare utility SETTTS and the set and get threshold function calls are provided to alter the number of locks required to start and end an implicit transaction. For example, some multiuser applications may always keep one or more records locked.

Implicit transaction tracking is designed to work transparently with existing multiuser software that uses record locking (physical or logical, NetWare or DOS). All the user must do is flag the multiuser database files as transactional; everything else is automatic. However, implicit transactions are not guaranteed to work with all multiuser applications. For example, applications that do not synchronize file updates exclusively with record locks and applications that synchronize updates improperly may not work.

Explicit transaction tracking has an advantage over implicit transaction tracking in that it allows applications to determine precisely when updates within the transaction are written to disk. In addition, NWTTSBeginTransaction and NWTTSEndTransaction calls within the application allow the developer to identify the beginning and end of file update sequences.

Identifying the beginning and end of file update sequences makes it easier for applications to group file updates and practice correct record locking practices. Correct record locking practices include locking and unlocking records as a group to avoid deadlock, and not leaving records locked exclusive while waiting for keyboard input.

Modifying an application with explicit transaction calls does not affect its operation in a non-NetWare environment.

Record Locking

Record locking provides security and data protection during transactions. If a record is not locked by an application but is written to during a transaction, the file server physically locks that record automatically. This physical lock remains in force until the transaction is completed. Physically locking written records is an added protection that prevents other workstations from examining or modifying them while they are being changed.

The file server usually holds logical and physical record locks on a file until the end of the transaction, even if the file is unlocked by a workstation before the transaction is completed.

For example, if a workstation requests an unlock before a transaction is completed, it will get a SUCCESSFUL completion code indicating that the records are unlocked; however, the lock is actually held until the transaction is complete. The one exception to this is a file that is not updated while the lock is in force. A request to unlock such a file is honored.

The file server delays record unlocking because the data controlled by the locks could still be changed by a transaction backout. Thus, the file server guarantees that other workstations will not see data that is being changed until those changes are final. If multiple stations attempt to change a file, only the first station is allowed to make the change. Usually, multiuser software synchronizes and prevents other workstations from examining records that are being changed.

If a workstation attempts to read from or write to a record that is physically locked by the file server, the workstation gets a ªlockedº error. This means that applications which do logical record locks can potentially get unexpected physical lock errors. However, because unlocking logical records is also delayed during a file write, this should never happen if the logical record synchronization is correct. The logical record locks keep other workstations away from the physically locked records.

It is important to point out how that in a non-TTS environment it is valid to unlock some updated records in the middle of a transaction if they have been completely updated. In a TTS environment, however, those records cannot be unlocked because they can still be changed--not by the application but by a transaction backout.

Transaction Backouts

Transactions are backed out because of system failures resulting from hardware problems and power outages at the workstation or the file server. But backouts also occur because of problems with applications running on a workstation or because of user intervention at a workstation.

Causes

Below is a list of some common causes for a transaction backout.

- Transaction backout occurs if an application terminates while a transaction is in progress (a begin transaction with no end transaction). For example, the user may enter a CTRL-Break.
- Transaction backout occurs if, after terminating, an application leaves records locked.
- Transaction backout occurs if there has been no activity from a station for 15 minutes. If a server does not receive packets from one of the workstations listed in its Connection Table for more than 5 minutes, it sends a watchdog packet to that station. If the station does not respond, the server continues to send a packet every minute for 15 minutes.
- After 15 minutes, if the workstation still does not respond, the server logs it out. Usually lack of response from a workstation indicates a power failure, software hang or a problem with an intermediate network route, etc.
- Transaction backout occurs if a station re-attaches to the same server after rebooting and reloading the shell. If the station attaches to a different server after a reboot, the file server watchdog process will re-initialize the station after 5 minutes.
- Transaction backout occurs if a CLEAR STATION or a DOWN command is issued at the file server system console.

Solutions

The following are some general suggestions of how to recover from a workstation or file server that goes down and therefore backs out a transaction.

Workstation

When a workstation goes down in the middle of a transaction the user may not know exactly which transaction was completed and which was backed out. However, unless the application is multi-tasking, it should have only one transaction active at the time of the crash. Even if a workstation goes down, transactions that were completed but not yet written to disk will be written to disk and will not be backed out.

If the station goes down and transactions were backed out, the user must ensure that the last transaction was complete, or if it was not complete, do it over again. It could be partially completed if the application incorrectly used several transactions per operation. The following message will appear at the file server console if transactions were backed out:

> ªTransaction being backed out for station ##.º

This message is given at the file server console if a transaction is backed out for any reason other than an Abort Transaction from the application. Note that if the station had several tasks active, several messages are given. The file server Down command can force several of these messages to be issued.

If a workstation goes down and cannot be brought back up, the user should issue a CLEAR STATION command at the file server console. Besides affecting the transaction tracking system, the downed station keeps its records locked, which may affect other stations. If the CLEAR STATION command is not issued, the file server watchdog process will clear out the station after 15 minutes.

File Server

Unlike the workstation, if the file server goes down before transactions have been written to the disk, it will back out any incomplete transactions when it is rebooted. If your file server does go down and files are backed out, the following message will be issued at the file server console:

> ª### transactions need to be backed out. Type a character to start the
> backout.º

This message will appear when the file server is being brought up (after all of the volumes have been mounted). The transaction tracking software has detected that some of the transactions were not completed before the server went down. You should not see this message unless the server went down abnormally.

After bringing up a file server that crashed, the user is responsible for determining which updates were completed and which were backed out and need to be repeated. A good practice is to verify that the last few changes were actually made. Remember that most transactions should be committed to disk within 3 to 5 seconds of being completed. However, any operation completed within 10 seconds of a failure should be examined. After the file server is brought up, a good practice is to have the workstation operators review their last operation. If it is not completed, they should also review the next to last operation, and so on.

If the NWTTSIsTransactionWritten call is used by the application, the user will have a better idea of when transactions were actually completed. However, just because the application thought that a transaction was not completed does not mean that it wasn't. The transaction could have been completed between the last time the application checked and the time the file server crashed.

Disable/Enable Transaction Tracking

Transaction tracking can be disabled or enabled from the file server console with the commands DISABLE TRANSACTIONS and ENABLE TRANSACTIONS. However, even when transaction tracking is disabled, tracking occurs in a partial way:

- The NWTTSBeginTransaction and NWTTSEndTransaction calls still denote transactions and work correctly.
- The NWTTSIsTransactionWritten call still returns correct status.

If TTS is disabled, the only loss to explicit and implicit transaction calls is the capability of backing out incomplete transactions. For example, if a file server or workstation fails while transaction tracking is disabled, you cannot back out because the backout information will not have been saved, and the database could be corrupted.

When transaction tracking is re-enabled at the system console, backout capability will only be provided to new transactions. Transactions that are in progress when transaction tracking is re-enabled won't have backout, and the possibility of file corruption still exists until these unprotected transactions are completed and written to disk. It is important to note that a transaction can be backed out if the workstation does not write to a transaction file after transactions are disabled. If a workstation does a write after transactions are disabled, the transaction is invalidated and any previous backout information for that transaction is ignored.

Disable Transactions

DISABLE TRANSACTIONS (NetWare v2.*x*), DISABLE TTS (NetWare v3.*x*) and NWTTSDisableTransactionTracking (API) are commands that turn off the backout capability of the transaction tracking system. Any transactions written after transactions have been disabled cannot be backed out. A transaction that has not written anything since transactions were disabled can still be backed out while transactions are disabled. Normally, this command is only used for testing.

Enable Transactions

ENABLE TRANSACTIONS (NetWare v2.*x*), ENABLE TTS (NetWare v3.*x*) and NWTTSEnableTransactionTracking (API) are commands that re-enabletransaction tracking. Any previous transaction backout information is erased, except those transaction backouts that were active at the time transactions were disabled and did not write anything. These transactions are not erased and can still be backed out.

Applications and TTS

Almost all applications should work correctly with TTS implicit or explicit transactions, or be easily adapted to do so. However, the following TTS record locking features discussed earlier could adversely affect certain applications:

- Physical and logical record locks remain in force until the end of the transaction.
- Records unlocked in the middle of a transaction are usually held until the transaction completes.
- The file server automatically generates a physical lock when a record is written if the record is not yet locked.

Explicit Transactions: Deadlocking

Consider, for example, the following application running without TTS: The application locks a ªheaderº record. It then needs to allocate data records in several files. It does this by locking an ªavailº record in one of the data files, allocates the record, changes the record and then unlocks the ªavailº record.

It continues in this manner, allocating another data record in another file, never locking more than one ªavailº record at a time. Having only one ªavailº record locked at a time is the way the application avoids deadlocks.

However, when the same application is run with TTS enabled, the file server keeps the ªavailº records locked (even after an unlock request) because the ªheaderº record is still locked and the transaction is still going. Meanwhile, the application thinks it has unlocked them.

Thus, one workstation locks ªavailº record A and another locks ªavailº record B. When the stations request an unlock on their ªavailº records, they will get back SUCCESSFUL completion codes even though the records remain locked. Then, when the first station attempts to lock ªavailº record B, and the second station attempts to lock ªavailº record A, the workstations will deadlock. An application that runs fine without transaction tracking cannot run with it.

It is important to note that the same types of deadlocking problems can occur if multiple stations attempt to extend the same transaction file. This is because the file server delays the completion of the second file extend until the first transaction is completed.

There are several ways applications can be changed to overcome these problems with transaction tracking.

- Deadlock can be avoided if the application always locks the ªavailº records in a certain order (ªavailº A, if ever locked, is ALWAYS locked before ªavailº B).
- Deadlock is also avoided if the NetWare ªlock record setº command is used to lock all ªavailº records at once.

Most multiuser application will not have deadlocking problems with TTS, because the cases that cause deadlock are obscure.

Implicit Transactions

Implicit transactions should work correctly with almost all multiuser software that uses record locks, except in the following circumstances:

Semaphore Synchronization

Multiuser packages that synchronize using methods other than record locks may not work with implicit transactions. For example, if an application uses semaphores or data in a file to synchronize, it may not correctly signal implicit transaction begins and ends to the file server.

Single or Multiple Transactions

The user and the application may view what are actually several transactions as a single transaction, while TTS treats the transactions separately. For example, suppose that to add a new customer, the user first adds the customer to the
ªbillingsº database files (one transaction), then adds the customer to the ªorder entryº database files (another, separate transaction). The user assumes there will be no problem adding to the ªbillingsº file and also to the ªorder entryº file.

However, if the file server (or workstation) goes down with the customer added to the ªbillingsº files and not to the ªorder entryº files, the user or the application may not detect this inconsistency; the database would remain inconsistent. Even if the problem is detected, the application may not allow the user to add the customer only to the ªorder entryº files, or delete the customer only from the
ªbillingsº file.

Premature Unlocking

Transactions should not continue after all locks are released because once all records in a transaction are unlocked, the application has lost control over the data. For example, an application may want to lock additional records to finish a transaction, but be unable to get the records locked. If this is the case, it should back out the changes it has already made. Even if an application could get additional records locked to complete a transaction, those records may have already been changed by another workstation.

Some applications may know that after they unlock all of their records, the rest of the transaction will always work and never have lock problems. However, these types of applications will only work with explicit transactions.

Transactions Too Large

If an application package never releases all of its record locks, everything is done as one large implicit transaction. It still works fine, but the granularity of the different updates is lost. Large transactions tie up disk space in the transaction work files. Having too many large transactions active could possibly fill up the volume containing the transaction work files. If this occurs, transaction backouts have to be disabled. Note that this type of package can also overflow the record locking tables, because most of the physical and logical record locks are not released until the transaction is completed.

If an application needs to keep a lock around forever (for example, to control the number of users the application supports), the solution may be to not flag the file with permanent record locks as transactional. Another possible solution is to add explicit transaction begin and end requests.

Applications, the User, and TTS

Novell is encouraging multiuser software developers to test, verify and certify that their software is TTS compatible. These developers may wish to publish a set of special instructions for running their package with TTS. TTS-aware applications can often minimize the user's need to understand TTS, but there are some things, particularly file server failure recovery, that require the user to understand how TTS works even with a TTS aware application.

The user also needs to understand how to handle transaction files. In particular, for example, these files cannot be deleted or renamed as explained in the introduction. Also, if the file is moved or restored from backup, the transaction attribute will be lost and will need to be restored.

It is also important for users to know that certain operations can be performed more efficiently without TTS. A good example is data file compression. Some databases require their data files to be compressed periodically to recover deleted records. The compression is performed by only one workstation and results in a new database being generated.

Before the compression, the user should flag the database files non-transactional so that all of the compression operations do not become one gigantic transaction. (Some database software uses file locks instead of record locks during compression, and will not have this problem.)

It is important for the user to understand the nature of compression and the fact that a new database file is generated. If application software were not TTS aware, the new database file would not be flagged by the application as transactional. The user must remember to handle this case.

The user needs to be aware of the TTS work file volume and its memory requirements.

The user needs to determine which files need to be flagged transactional. This implies the user knows the purpose of each application file. All shared database files including index files should be marked transactional. Failure to mark all files violates a fundamental assumption of the TTS system and will cause transaction backouts to be incomplete.

The user needs to know how to operate the FLAG utility.