TANDEM

Introduction to Transaction Monitoring Facility (TMF)

Tandem NonStop^{TM}

INTRODUCTION TO THE TRANSACTION MONITORING FACILITY (TMF)

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, California 95014

Product No. 8063 A00
Part No. 82063

October 1980
Printed in U.S.A.

First printing December 1980
Second printing February 1981

PREFACE

ENCOMPASS—Tandem's Distributed Data Management System—contains five products: ENSCRIBE, ENFORM, DDL, PATHWAY, and the Transaction Monitoring Facility (TMF). This publication describes TMF. The purpose of the publication is to introduce TMF concepts and to provide preliminary information for users planning to install TMF.

The information in this publication is organized to allow you to make a simple transition from learning about TMF to understanding the principles related to installing and using TMF. It contains five sections:

1. Overview
2. Concepts
3. TMF Operations
4. TMF Feature Description
5. Using TMF

This publication is written for system managers, application programmers, and Tandem analysts. Depending on your specific job, it is suggested that the publication be used as follows.

- All users should read the Overview section for a general explanation of TMF features, the problems that it solves, and its relationship to the features provided by the GUARDIAN operating system.

- Users interested in background detail for concepts related to TMF such as transactions, database consistency, and concurrency should read the Concepts section.

- All users should read the TMF Operations section to understand how TMF uses transactions to maintain database consistency.

- Individuals responsible for installing and controlling TMF (system managers) should read the TMF Feature Description section to understand the relationship between the TMF system processes that they will be responsible for configuring and the TMF features.

- All users should read the Using TMF section for information related to configuring TMF, programming in a system with TMF, and operational characteristics of TMF. This section contains planning information that is intended to highlight the major tasks involved in configuring and using TMF.

This publication is intended for planning purposes. Detailed reference information for using TMF is contained in the TMF Users Guide that will be available with the release of TMF.

See the following publications for information describing the ENFORM, DDL, PATHWAY, and ENSCRIBE components of ENCOMPASS.

- ENFORM reference manual.
- DDL Data Definition Language Programming manual.
- PATHWAY Reference manual.
- ENSCRIBE Data Base Record Manager Programming manual.

CONTENTS

LIST OF FIGURES

SECTION 1

OVERVIEW

The Transaction Monitoring Facility (TMF) is a product provided by Tandem as part of its ENCOMPASS Distributed Data Management System. The purpose of TMF is to simplify the task of maintaining database "consistency". But, what is "consistency" and why is it important? One way to think about it is to first view consistency as part of a larger problem that you encounter every day. The problem—maintaining database integrity—is to ensure that facts about your business or profession, represented by items in a database, reflect your business or profession correctly.

Clearly, maintaining database integrity is not simple. But TMF simplifies one related concern: that is, when you design a database, you establish criteria for the relationships between items in the database; for example, an account balance equals its credits minus its debits. When the database satisfies the criteria, it is consistent. And for your database processing applications to work correctly, they must be able to view their input from the database as consistent regardless of the changes made to the database.

THE GUARDIAN OPERATING SYSTEM AND ENCOMPASS

Figure 1-1 illustrates the relationship of the GUARDIAN operating system to TMF and two other ENCOMPASS products, ENSCRIBE and PATHWAY.

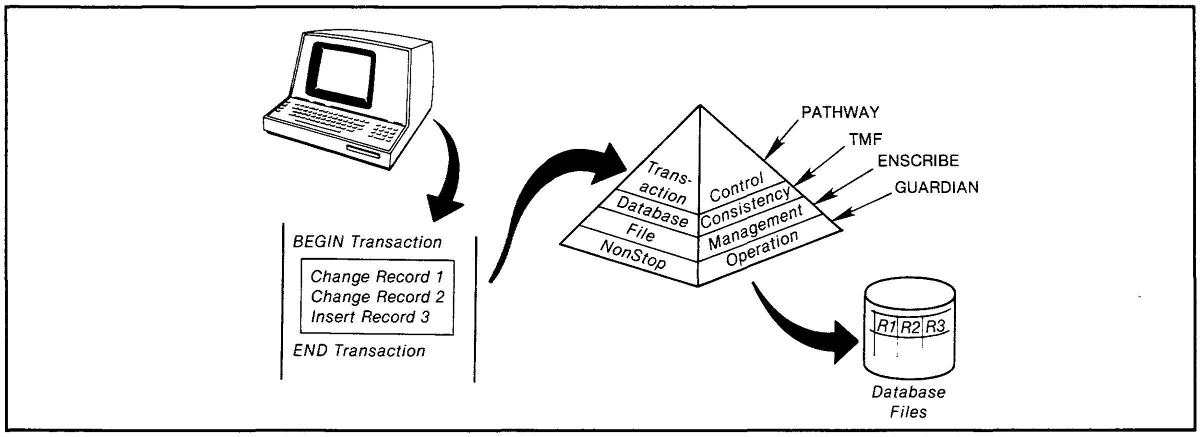


Figure 1-1. Relationship of GUARDIAN to ENCOMPASS

There are several important concepts in Figure 1-1:

- PATHWAY provides a simplified terminal-oriented interface for transaction design and control.

- TMF maintains consistency for the databases changed by the transactions.

- ENSCRIBE provides the file management required to access and change the databases.

- GUARDIAN provides a reliable environment to ensure that PATHWAY, TMF, and ENSCRIBE operations proceed regardless of single component failures.

The key concept in Figure 1-1 is the "transaction" as a multi-step operation that changes the database. The changes transform the database from one consistent state to a new consistent state, but generally make it inconsistent at points during the transformation. For example, a transaction that is part of a banking application balances an account by a withdrawal operation and a credit operation. The state of the database will be inconsistent after the withdrawal but before the credit; it violates the criteria for balancing accounts.

To maintain consistency then, TMF ensures that all of the changes that a transaction attempts to make are permanent or that none of the changes are permanent. In either case, the effect on the database is that its consistency is preserved; it will not reflect the results of any partial transactions.

The start of a transaction is identified by the BEGIN-TRANSACTION verb. The END-TRANSACTION verb is used to indicate that the transaction is complete and its changes to the database should be "committed". Committed simply means that the transaction's changes are permanent; they will not be undone by TMF. If a transaction is aborted for any reason before END-TRANSACTION or before its changes are committed, TMF will back out the changes; i.e., restore the database to its initial state. The transaction can then be restarted from the beginning.

INTRODUCING TMF

The GUARDIAN operating system provides several features to maintain integrity: continuous availability, file structural integrity, storage media protection, data sharing protection, and application-level NonStop programming. All of these features help to ensure that changes to the database occur successfully in spite of single component failures.

Now, with the addition of TMF, the Tandem philosophy of failure protection has been extended to solve the specific problem of maintaining database consistency if: (1) a transaction fails (is aborted) before its changes are committed and (2) a total system failure (one not recoverable by normal NonStop mechanisms) occurs while transactions are changing files affected by the failure.

With TMF, application programmers do not have to be concerned with the following problems:

- Concurrency ... several transactions are applying changes to the database at the same time, but if a programmer follows record locking rules imposed by TMF, he can think of his transaction as the only activity in the system.

- Failure ... all of a transaction's changes occur successfully or none of them occur. Once a transaction ends and commits, its changes persist in the database despite any subsequent failure. In a PATHWAY environment, not having to worry about single component failures eliminates the need to program NonStop servers.

- Location ... TMF will maintain consistency across all nodes of a network distributed database regardless of where the transaction changing the database originates or how it is geographically distributed.

Note that TMF ensures the consistency of the database in terms of the transactions that change it. However, whether or not the database is a correct model of the application world depends on other factors, such as the correctness of application logic, the transaction definition, and the input data. For example, there is no good way of verifying that an input value of 12 gross of pencils is wrong if only 11 gross were received. Each of these factors should be taken into consideration when designing the applications and database.

TMF Feature Summary

Database consistency is maintained by TMF system-level code that provides five features: (1) Audit Trails, (2) Transaction Backout, (3) Transaction Concurrency Control, (4) Online Dump, and (5) Rollforward recovery from total system failures. The features are briefly described in this section. For more detail, see the TMF Operations and TMF Description sections of this publication.

AUDIT TRAILS. Audit trails are groups of files that TMF uses to record information about a transaction's changes to a database (Figure 1-2). Information in the audit trails includes status information about a transaction such as whether it completed successfully, and before and after images of all records changed by a transaction. The before images are copies of the database records before the transaction changed them; they are used to back out an aborted transaction. The after images are copies of the changed records; they are used to restore the database to a consistent state in the event of a total system failure. Audit records for a transaction are written to the audit trail disc before the transaction commits.

If the database is distributed over nodes of a network, separate audit trails are maintained on each node. Before and after images of changes to a database file on a node are written to an audit trail on the same node.

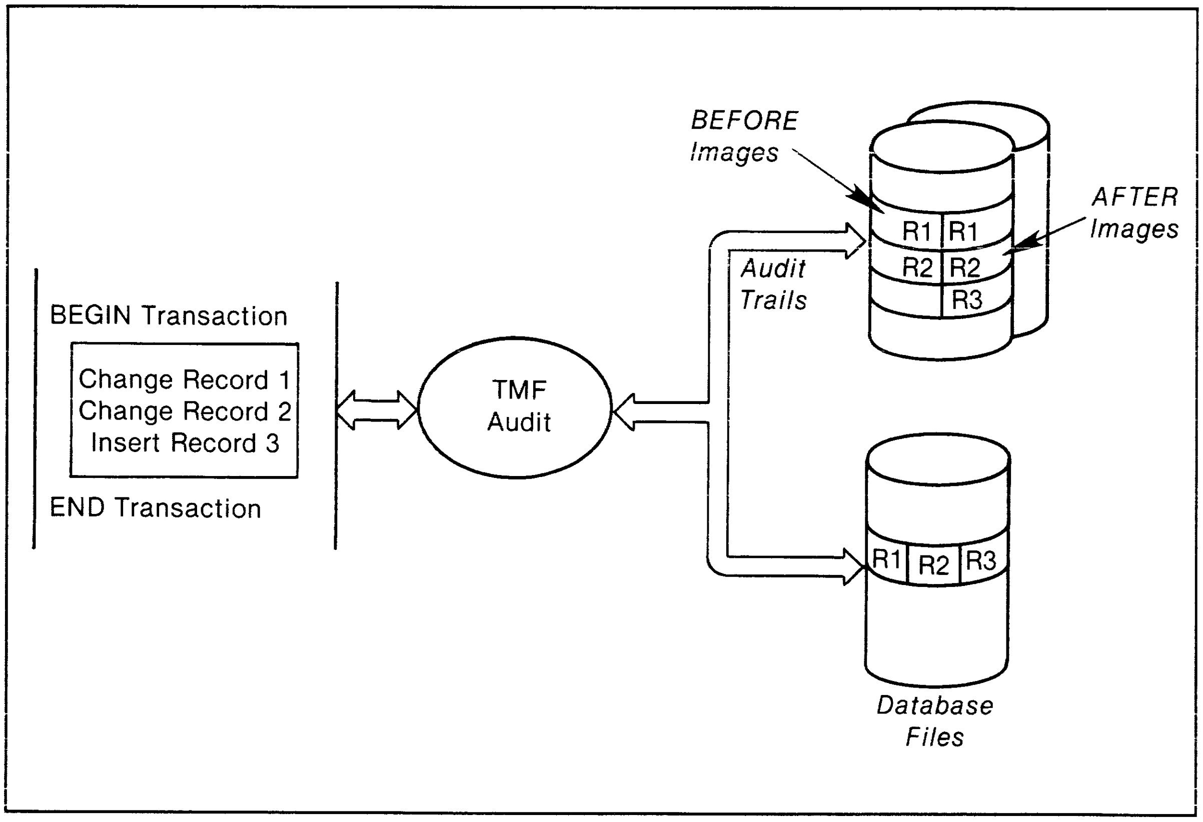


Figure 1-2. Audit Trails

TRANSACTION BACKOUT. Transaction backout removes the effects of an aborted transaction (Figure 1-3) from the database by using the before images in the audit trails to undo all of the transaction's changes. If the transaction affects data distributed over a network, backout occurs independently at each network node. Essentially the result of backing out a transaction is the same as if the transaction's changes had never occurred anywhere.

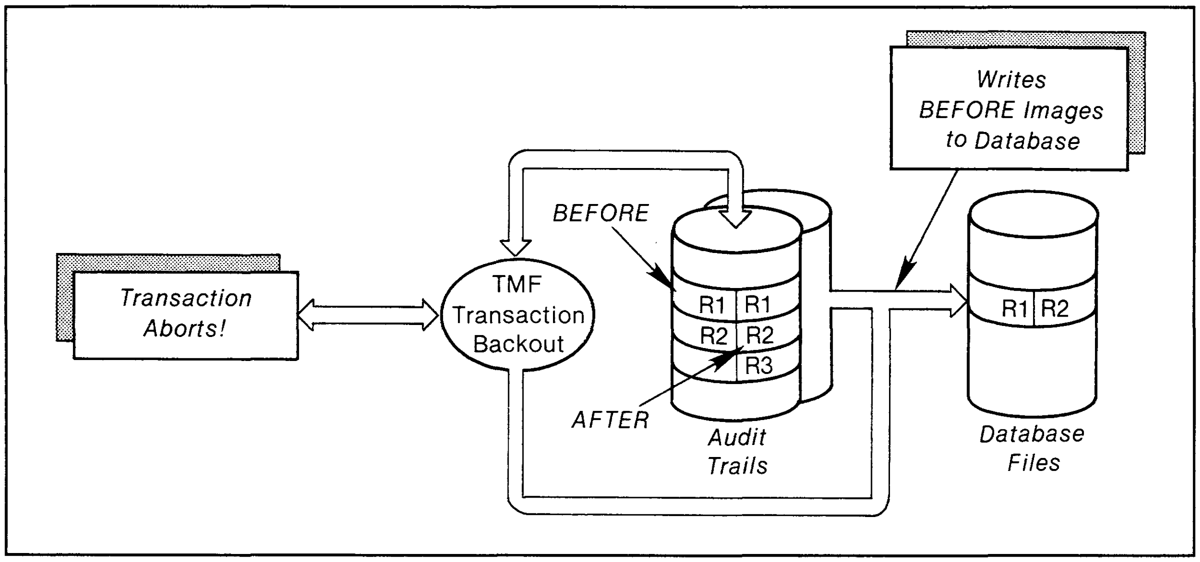


Figure 1-3. Transaction Backout

TRANSACTION CONCURRENCY CONTROL. TMF concurrency control (Figure 1-4) ensures that all changes made to a database during a transaction's processing are not visible to other concurrent transactions until the transaction's changes are committed. Concurrency control depends on a record locking mechanism, enforced by TMF, that ensures that all records modified, deleted, or inserted by a transaction are locked until the transaction commits.

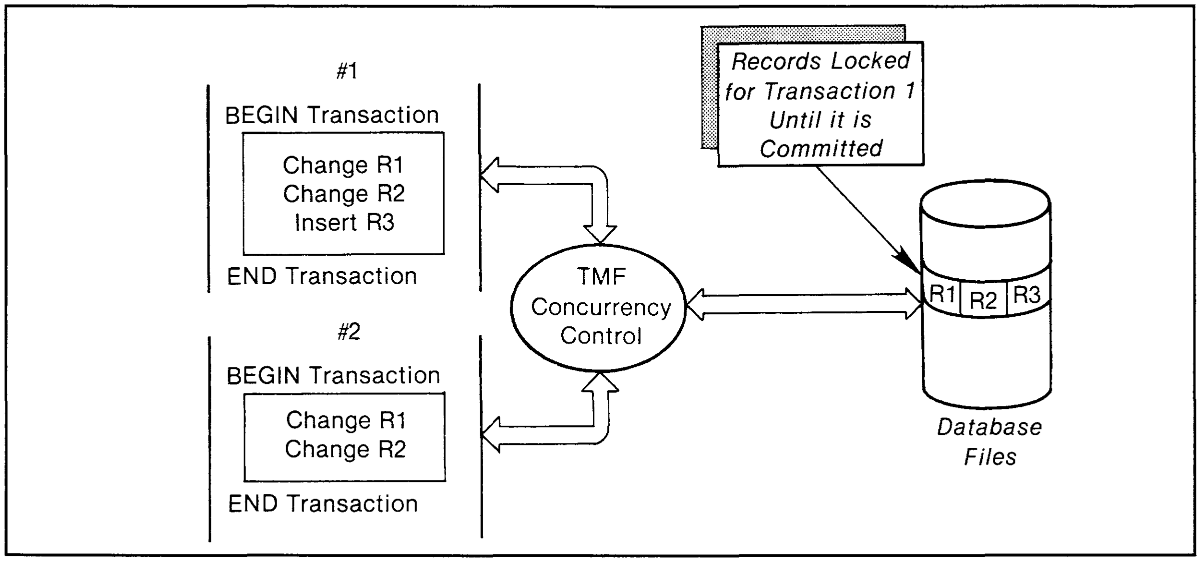


Figure 1-4. Transaction Concurrency Control

ONLINE DUMP. Online dump is used to copy selected portions of a database (Figure 1-5) to tape, preserving them for future use in the event that they need to be recovered after a total system failure. Online dump can be run while the database is being updated.

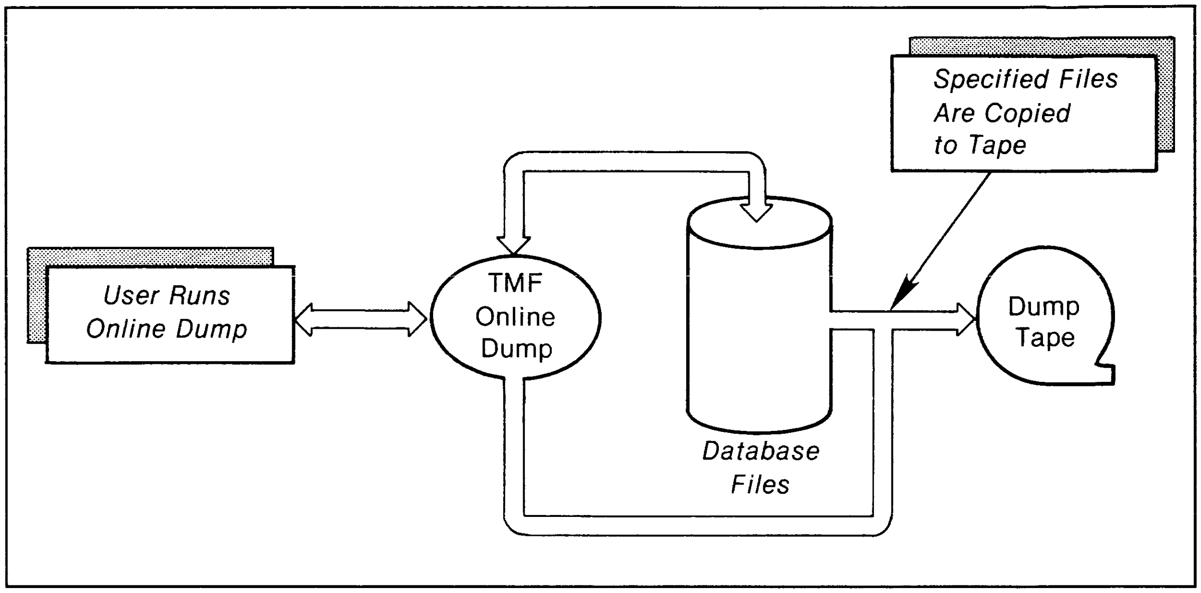


Figure 1-5. Online Dump

ROLLFORWARD. Rollforward is used to recover (to the most recent consistent state) files that were open and being changed by transactions when a total system failure occurred (Figure 1-6).

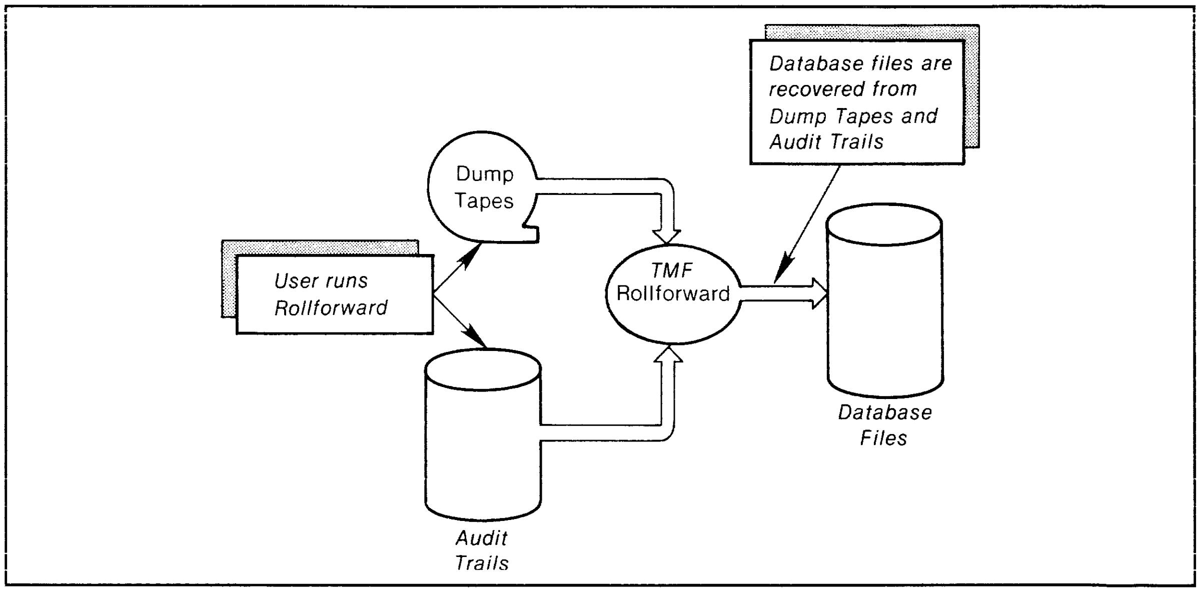


Figure 1-6. Rollforward

DATABASE CONSISTENCY AND FAILURES

TMF will maintain the consistency of transaction input from the database regardless of the following types of failures or problems.

1. Single component failures that may cause a transaction to be aborted, but are non-critical events because the fault-tolerant features of GUARDIAN enable processing to continue in other components. An example of a failure in this category is failure of the cpu where the transaction is processing.

2. The transaction is voluntarily aborted by the application.

3. Total system failures caused by:

   - Any two hardware failures in the same data path.

   - Power failure (blackout or possibly brownout).

   - Human error.

For aborted transactions, the audit trails are used to back out the transactions by writing the before images of all records changed by the transaction back to the database. In a system with PATHWAY, the aborted transactions are automatically restarted.

Audit trails are not affected by single component failures because the system processes that write and maintain them are NonStop and they are on mirrored disc volumes. (A mirrored volume is a pair of physically independent devices that are accessed as a single volume and managed by the same input/output process.)

For total system failures, the Online Dump tapes and the audit trails are used to restore the database to a consistent state. This occurs as follows:

- The database files affected by the failure are restored from the Online Dump tapes.

- The after images of all transactions that committed since the time of the last Online Dumps for the files open at the time of the failure are written to the files from the audit trails. The after images of transactions that aborted or were incomplete at the time of the failure are not applied to the database.

TMF SYSTEM REQUIREMENTS

Using TMF requires the following system and hardware components.

- The T/16 GUARDIAN operating system with ENSCRIBE.
- Sufficient disc space for the audit trails.
- Sufficient tapes and tape drives for dumping the database files.

SUMMARY OF USER RESPONSIBILITES

Three groups of users are affected by TMF: application programmers, system managers, and operators. Their responsibilities are briefly described below and in more detail in the Using TMF section of this publication.

Application Programmers. TMF can be used by TAL, COBOL, FORTRAN and PATHWAY programmers. Programmers have to decide how to design their applications to use transactions; identify the beginning and end of each transaction; and lock any data that will be changed by their database processing servers.

System Managers. System managers are responsible for ensuring that the system facilities required to support TMF are adequate. Their responsibilities include:

- Determining and specifying which files in a database should be audited by TMF.
- Using SYSGEN and the TMFCOM utility (a new utility provided by TMF) to configure TMF and to specify the characteristics of the audit trails used by TMF. Characteristics that can be controlled by the user include: the number of audit trails, the amount of disc space allocated to them, and their location.
- Determining how often Online Dump should be used.
- Using TMFCOM to control TMF.

Operators. Operators are responsible for:

- Running Online Dump and mounting the tapes that it requires.
- Starting Rollforward after a total system failure.
- Mounting the tapes required during Rollforward.

SECTION 2

CONCEPTS

This publication has no glossary and unique terms are generally defined as they are introduced. However, some concepts related to TMF are essential to your understanding of the information presented in the following sections and are described in detail here. They are:

- Databases and database consistency.
- Transactions.
- Distributed database systems.
- Data sharing and transaction concurrency.

DATABASES

The general definition of a database is

> "Stored information that is: (1) subject to concurrent multiple user access and updates and (2) viewed by application programs as a model of the real world."

Within this general definition, two items related to TMF are important:

1. Stored information is data in a set of related disc files that your applications are concerned with.

2. The state of the real world represented by the database is continuously changing, but each transaction must get consistent input from the database regardless of the changes.

Database Consistency

Figure 2-1 illustrates the stock files and the relationships between them for a distributorship (an organization that receives parts from suppliers and resells them to customers) database. Each part in the PARTS file is supplied by one or more suppliers.

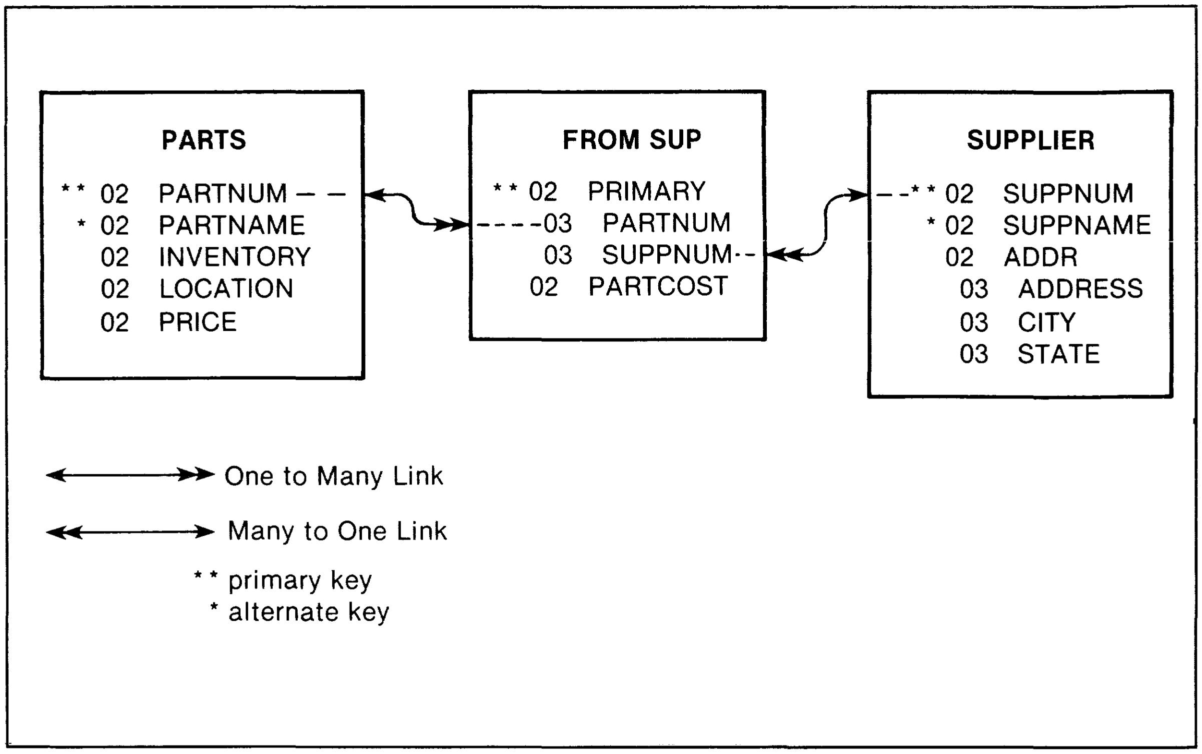


Figure 2-1. Stock Files

Your design criteria for relationships between the files can be expressed as assertions about the data they contain. When the assertions are satisfied, the database is consistent. For example, in the stock files:

- Every record in the FROMSUP file uniquely identifies a record in the PARTS file.
- Every record in the FROMSUP file uniquely identifies a record in the SUPPLIER file.
- Every record in FROMSUP indicates who currently supplies a particular part.
- The alternate key file correctly indexes the primary file on the alternate key field SUPPNAME.

These assertions must be true for the distributorship database to be useful. If a transaction deletes a supplier record from the SUPPLIER file, it must also delete every record occurrence of the same supplier from the FROMSUP file. But, if the transaction making the deletions is aborted before completion and the changes are not backed out, the distributorship database will be inconsistent. Other transactions using the files might see relationships that are not true. This is the kind of problem that TMF is designed to prevent by backing out partial transactions.

TRANSACTIONS

Applications commonly make a series of changes to a database while transforming it from one consistent state to another. During the transformation, some of the intermediate database states will fail to satisfy one or more consistency assertions. For example, a banking application withdraws money from one account and deposits it in another by updating two records. The state of the database will be inconsistent after the withdrawal operation but before the deposit; it violates an assertion about balancing accounts.

Conceptually and practically, it is useful to have a way of identifying the series of changes that transform the state of a database. This leads to the idea of a transaction as "a group of changes to a database that transform it from one consistent state to another, but generally leave the database in an inconsistent state during the changes." Transactions are bounded—they have a beginning and an end—and they are manipulated by TMF as units.

Transactions should be distinguished from the applications that define the sequence of operations in a transaction. With TMF, application systems identify the start of transaction execution by issuing begin transaction statements and terminate transaction execution with end transaction statements. The begin and end transaction statements identify the transaction as an active unit and the transaction either modifies the state of the database or it fails, is backed out and has no effect on the database.

The concept of a transaction is the key to how TMF provides all of the features discussed in the Overview section. Once a transaction is identified, TMF uses its changes as a unit of recovery for backout and total system failure.

Note that your application may view the transaction as a logical unit of work; for example, entering the order header along with all of the detail items in a purchase order. However, TMF treats the transaction as a physical unit of recovery. When you design your applications, you have to consider this difference. Basically this means deciding if logical units such as purchase orders will be subdivided into a number of smaller recovery units or kept together in one large recovery unit. See the Using TMF chapter of this publication for a more detailed description of transaction design.

DISTRIBUTED DATABASE SYSTEMS

Distribution refers to the location of a database or to the location of processing affecting a database. A "locally distributed" database resides on more than one disc volume within a single Tandem system. A "network distributed" database resides on more than one node of an EXPAND network where a "node" is a Tandem system. Nodes may be geographically dispersed. "Network distributed transactions" are transactions that change parts of a network distributed database. Figure 2-2 illustrates distribution (local and network) on a Tandem system.

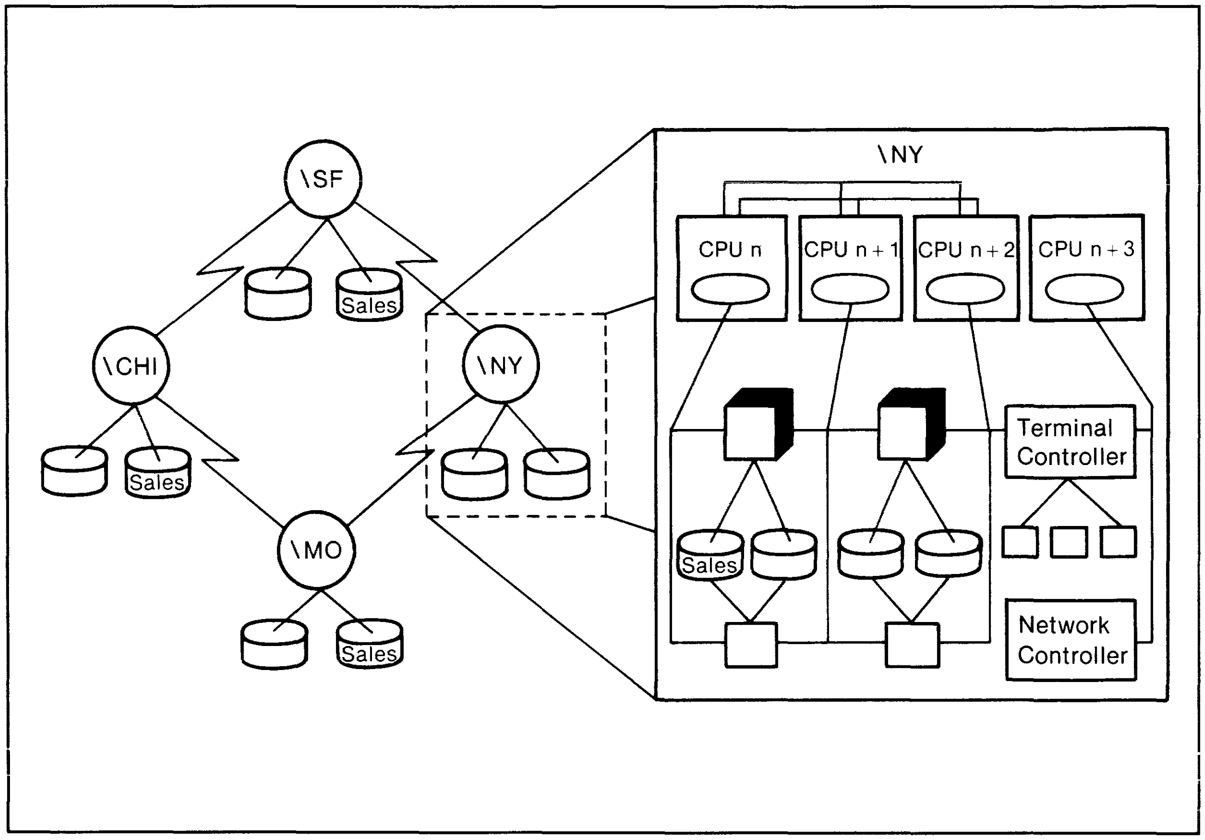


Figure 2-2. Distributed Database Processing

The "home node" is the node where the transaction begins. TMF operations take place at each node where the distributed database resides, regardless of where the transaction changing the database was begun. Audit trails are local to each node and all TMF operations related to transaction backout and Rollforward recovery occur at the individual nodes.

The TMF features that maintain consistency for distributed databases are transparent to application programmers.

DATA SHARING AND TRANSACTION CONCURRENCY

The definition of a database included the idea that a database is designed to be used by different application programs. The definition implies that an important characteristic of a database is that its contents are shared: different application programs will attempt to use the same entities in the database.

Sharing data does lead to some problems that potentially affect the consistency of the database. Specifically there is a difference between sequential execution of transactions and concurrent execution of transactions, and the difference affects database consistency.

If transactions run one at a time in some sequential order determined by the system, each transaction sees a consistent database state that reflects the results of the last completed transaction. However, sequential processing is generally impractical in a shared environment and transactions usually run concurrently: several different transactions are attempting to read and/or change the same database at the same time (Figure 2-3).

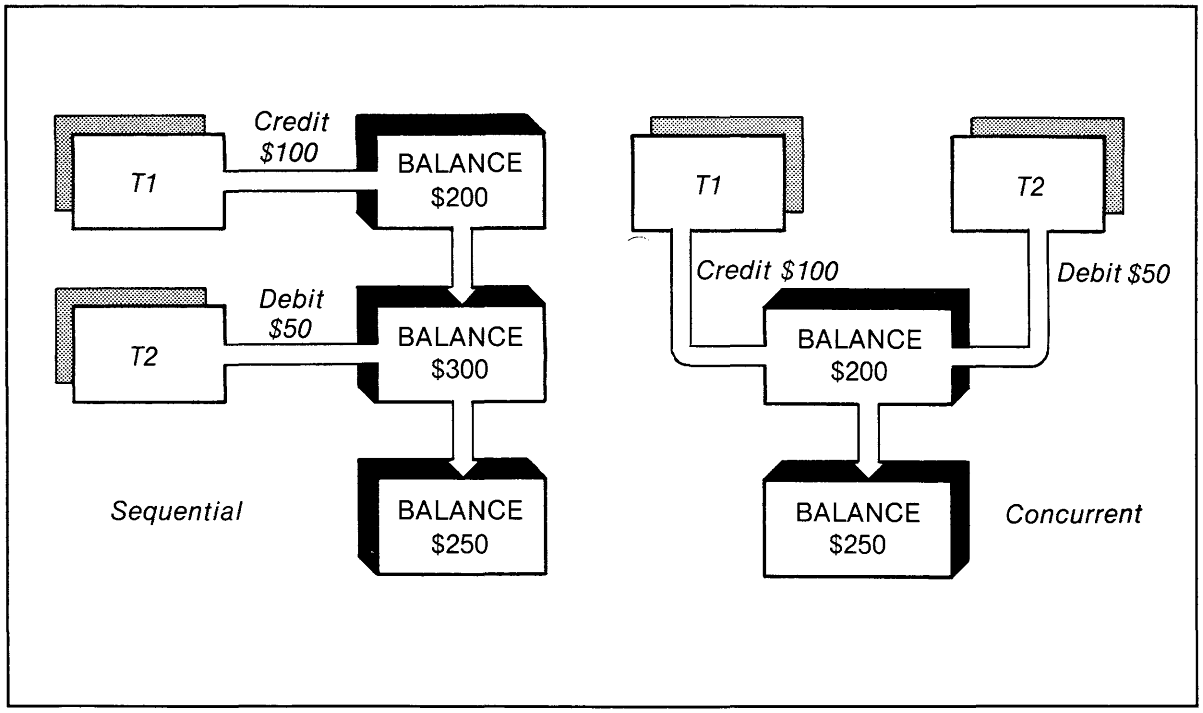


Figure 2-3. Concurrent Vs. Sequential Transactions

Since a transaction causes a database to become temporarily inconsistent, it is important to have a mechanism that screens the changes made by a transaction from other concurrent transactions until all changes have been committed. Otherwise the consistency of the database could be affected by the interaction between transactions and may lead to the following types of problems:

- Two transactions read and change the value of some entity and concurrent execution causes the value to be incremented only once. For example, in the banking application, suppose that the account initially had a $200 balance and one transaction attempts to credit $100 and another attempts to debit $50. If the transactions run sequentially, the balance will be $250. If the transactions run concurrently, they might both update the balance to give a balance of $300 or $150 depending on which transaction did the last update to the database (Figure 2-4); one of the updates would be lost.

- If transactions can read output from transactions that are still liable to backout, they see database states that could not be observed in a non-concurrent system. For example, if a transaction attempts to read two accounts, A1 and A2—one being debited, one being credited by another transaction—it may see a situation where money has disappeared: A1 debited but A2 not yet credited.

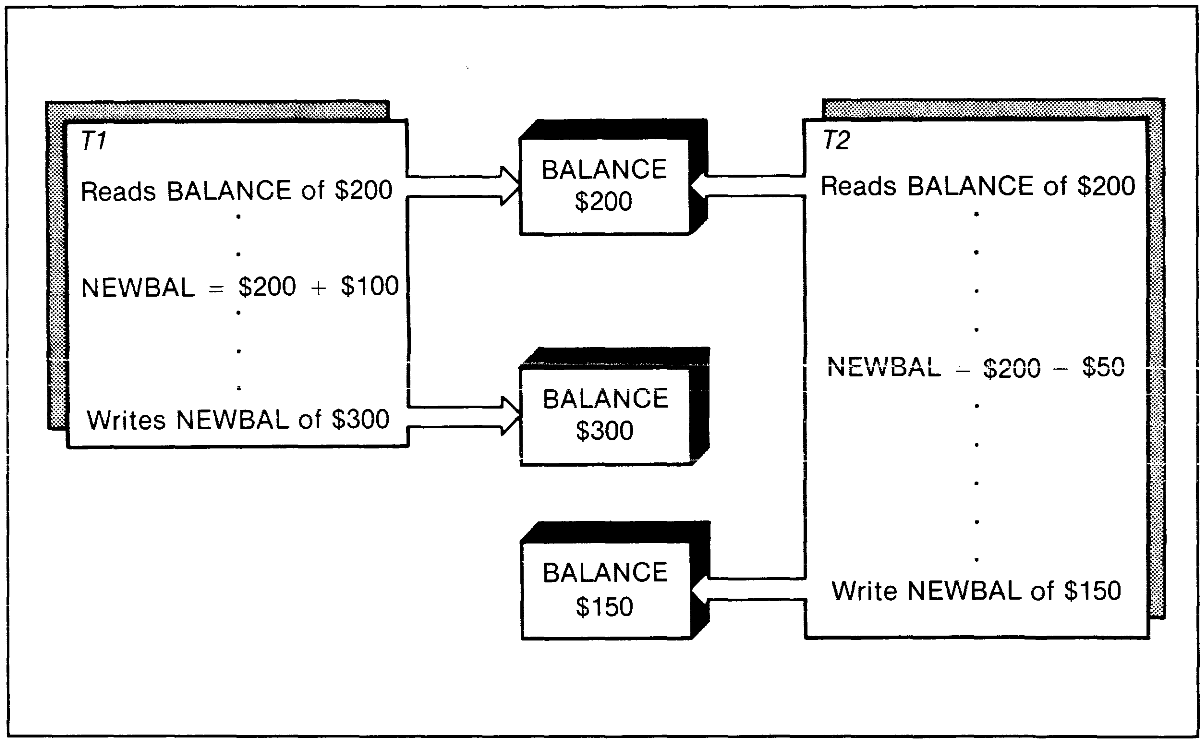


Figure 2-4. Transaction Concurrency—Lost Update Problem

TMF's concurrency control mechanism prevents the preceding problems by: (1) ensuring that all records changed by a transaction are locked and (2) holding the locks until the transaction ends and is committed. TMF's concurrency control prevents transaction dependency. A transaction can be backed out individually without affecting other concurrent transactions.

With TMF then, concurrent execution of transactions is equivalent (in terms of database consistency) to sequential execution.

TRANSACTION COMMIT. The term "commit" refers to the point in time when a transaction's changes are permanent. To summarize, think of it as follows: before changes are committed, they can be rewritten by the transaction or backed out by TMF; after changes are committed, they cannot be backed out, but can be recovered by TMF in the case of total system failure; and finally, TMF prevents transactions from seeing uncommitted changes produced by other concurrent transactions.

SECTION 3

TMF OPERATIONS

In the Concepts section, the transaction was described as the key to the TMF operations that maintain database consistency. This section will expand that idea by tracing the history of a transaction through a system with TMF (Figure 3-1) and show how:

- A transaction is originated and identified as an active recoverable unit.
- Transaction identifiers are transmitted between processes that cooperate to do the work of the transaction.
- TMF prevents transaction interaction that could result in transaction dependency by enforcing record locking rules.
- A transaction is committed.

The programming-related information in this section generally applies to systems with PATHWAY; it assumes a terminal (i.e. a Screen Cobol program executing on behalf of a terminal) issuing the BEGIN-TRANSACTION and END-TRANSACTION verbs that define a transaction. For information on using TMF outside a PATHWAY environment, see the Using TMF section of this publication.

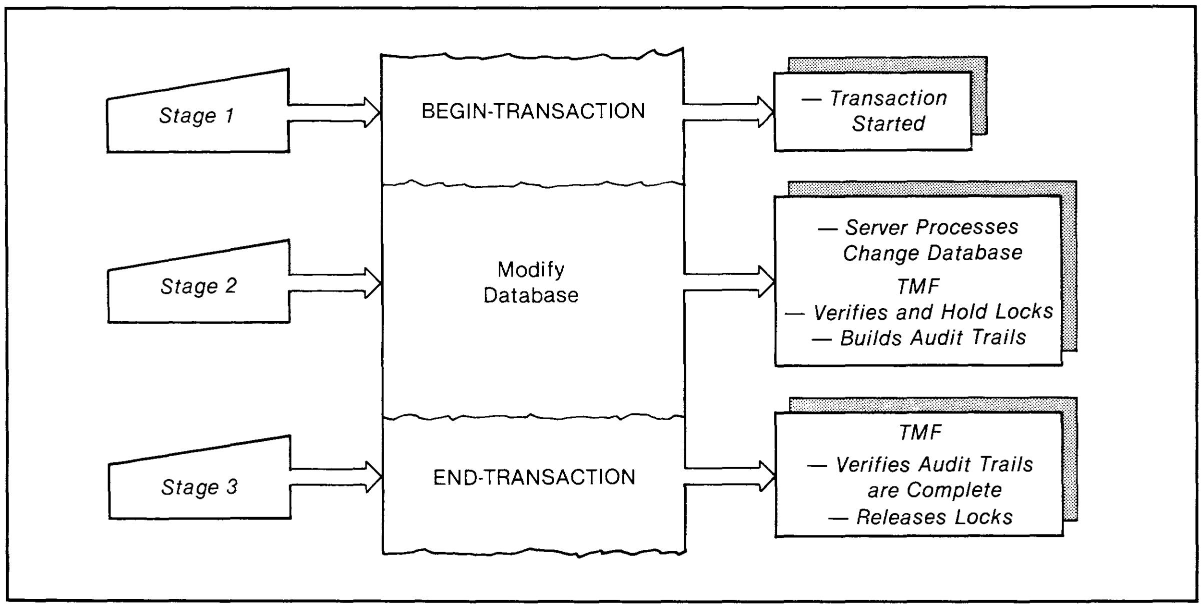


Figure 3-1. TMF Operations

Conceptually, Figure 3-1 illustrates that a transaction's changes occur in three stages.

Stage 1	the Screen Cobol program issues BEGIN-TRANSACTIONand a new transaction is created.
Stage 2	in this stage, the transaction modifies the database by reading, changing, deleting, and inserting records. During this stage, a transaction can rewrite any records that it changed. Concurrently, TMF is building an audit trail of all records inserted, deleted, and changed by the transaction and ensuring that all records are locked.
Stage 3	the Screen Cobol program issues END-TRANSACTION to indicate that its changes should be permanent. TMF will:

- ensure that the audit trail is complete for the transaction's changes; all changes to all files are in the audit trail so that they can be used to recover the transaction after a total system failure.
- release the transaction's locks.

TRANSACTION IDENTIFICATION

In a Tandem system with TMF, a transaction is a uniquely-identified entity known to the system. Each transaction is distinguished from other transactions by a transaction identifier (TRANSID).

A PATHWAY programmer identifies the beginning of a transaction with the BEGIN-TRANSACTION verb in the PROCEDURE division of his Screen Cobol program. When this verb is executed, a new TRANSID is originated; the terminal is switched to transaction mode; and the new TRANSID is associated with the terminal.

TRANSACTIONS AND PROCESSES

To understand transactions, it is useful to relate them to the idea of a process in a Tandem system. A process is the unique executing entity that is created when someone runs object code from a program file. More generally, a process can be thought of as a running program. Each time a user requests program execution, a process is created and, internally, each process consists of:

- An unmodifiable code area that contains machine instructions.
- A separate private data area called a stack.
- An entry in a system table called the Process Control Block (PCB) that uniquely names the process to the system by its cpu number combined with its process identification number.

The execution of a transaction can involve several processes and the PCB has been expanded to include space for the identity of the transaction that is currently active in the process: the current process TRANSID. The current process TRANSID uniquely identifies the active transaction for an active user process in the system.

After a Screen Cobol program has initiated a new TRANSID by issuing BEGIN-TRANSACTION, two types of processes generally cooperate to accomplish the changes required by the transaction: the Terminal Control Process (TCP) and server processes (Figure 3-2). The processes communicate using messages. And to ensure that all the work for a transaction is identified with a specific TRANSID the TRANSID is attached to all messages sent by the processes. The TRANSID is copied into the PCB of the receiving process; it then becomes the current process TRANSID for the process.

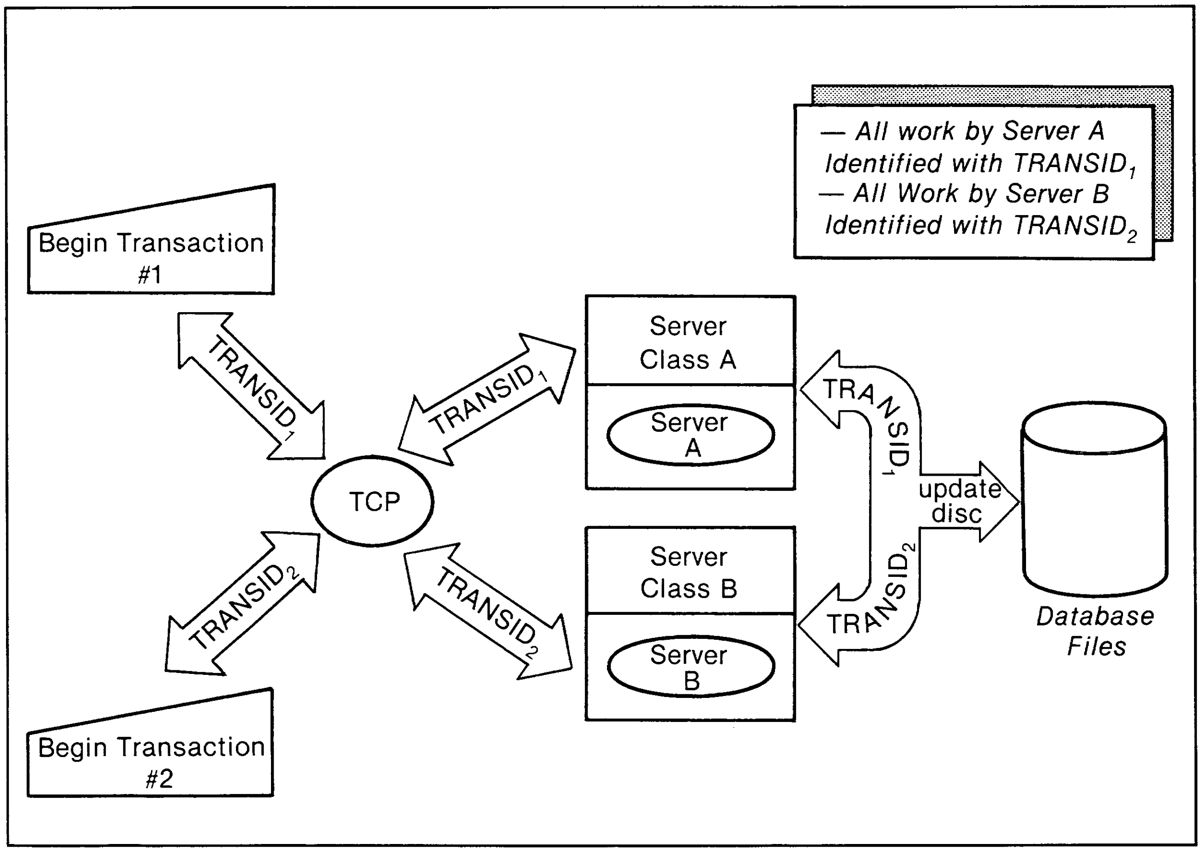


Figure 3-2. Relationship Between TCP and Server Processes

Each TCP interacts with one or more terminals to support the overall processing flow of each terminal by handling messages from several terminals concurrently and sending them to the appropriate server process. One transaction can result in a multi-step update that requires several servers. Each server involved in the transaction is then identifed with the TRANSID of the Screen Cobol program that issued BEGIN-TRANSACTION.

Each server process implements a specific application function involved in changing the database. The server picks up the request message from the TCP by reading a file called $RECEIVE and then typically satisfies the request by performing some database operation. All locks obtained by the server will be owned by the TRANSID. This allows multiple servers to do work for the same TRANSID. After satisfying the request, the server replies with a message indicating the disposition of the request.

Transmitting a Transid Between Processes

When a server reads $RECEIVE to pick up the request message, it automatically acquires the identity of the current process TRANSID passed in the message. The acquisition of the current process TRANSID is transparent to the server; it is handled by the file system.

Transmitting The Transid Over The Network

A current process TRANSID is transmitted over the network when either of the following occurs:

- I/O on a disc file that resides on a remote node.
- Interprocess I/O to a server running on a remote node.

When a transaction is transmitted to a remote node, it is begun on that node. All work done for the transaction on the remote nodes will be identified with the TRANSID originated on the home node of the transaction.

To enable the transaction to commit or abort on all nodes, a TMF system process (called the Transaction Monitor Process) on each participating node records: the nodes that sent it a transaction—its source node(s)—and any nodes that it sends a transaction to—the destination nodes for the transaction (Figure 3-3).

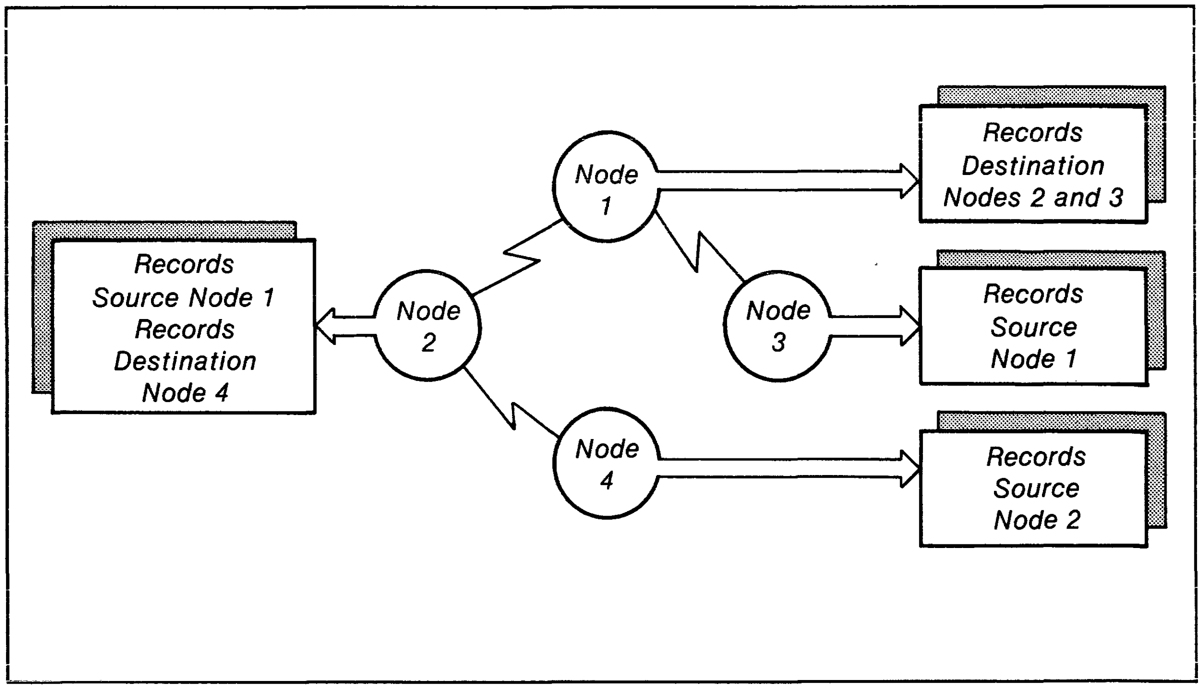

Records Source Node 1 Records Destination Node 4
Node 2
Node 1
Records Destination Nodes 2 and 3
Node 3
Records Source Node 1
Node 4
Records Source Node 2

Figure 3-3. Distributed Transaction—Transmittal

PREVENTING TRANSACTION INTERACTION

When you write server processes that change a database, you must follow a record locking discipline that TMF imposes to prevent transaction interaction. The discipline prevents transactions from reading uncommitted changes of other concurrent transactions.

The discipline consists of the following rules:

1. A transaction must lock all records that it updates. Specifically, this means that before a server process can successfully change or delete an existing record, it must previously have locked the record. Otherwise, the attempt to make the change or deletion will fail. Locks can be acquired on a record-by-record basis or for an entire file at a time. All locks are associated with the current TRANSID of the process at the time that it issues the lock request. Since lock owners are identified by their TRANSID, different processes that do work for transaction can lock records in one process and change them in another.

2. For records deleted by a transaction, TMF sets a new type of lock called a "key lock" that effectively prevents any other transaction from inserting a record at the key value of the deleted record for the duration of the transaction that deleted the record. All locks are held until the transaction completes.

3. TMF locks all new records inserted by a transaction and holds the locks until the transaction ends.

4. At the discretion of the programmer, all records read and not changed, but used by a transaction in producing its output should be locked. Following this rule guarantees that all reads are repeatable.

Rules 1-3 are enforced by TMF; rule 4 is not enforced.

TRANSACTION COMMIT

PATHWAY programmers identify the end of a transaction with the END-TRANSACTION verb in their Screen Cobol program.

When END-TRANSACTION is executed, TMF uses a mechanism called Two-Phase Commit to ensure that the changes are permanent (recoverable in the event of a total system failure) before releasing the record locks held for the transaction. If the transaction is network distributed, TMF ensures that all of its changes can be committed at all nodes touched by the transaction.

Two-Phase Commit

Two-phase commit for a locally distributed transaction is:

Phase 1 ...	phase 1 forces all before and after images associated with the TRANSID being committed, to be written to the audit trails. When a transaction reaches this state, a "transaction commit record" is written to the audit trail. The transaction commit record indicates that the committed transaction's changes are permanent and can be recovered after a total system failure.
Phase 2 ...	phase 2 is the releasing of all locks associated with the TRANSID.

For network distributed transactions, the same two phase-commit mechanism is followed with the addition of communication between the nodes to ensure that all nodes involved in the transaction are able to commit the transaction.

During phase 1 for a network-distributed transaction, a TMF system process (the Terminal Monitor Process) on the home node for the transaction sends a message to each destination node participating in the transaction. The message requests the destination nodes to perform phase 1 of the commit (Figure 3-4).

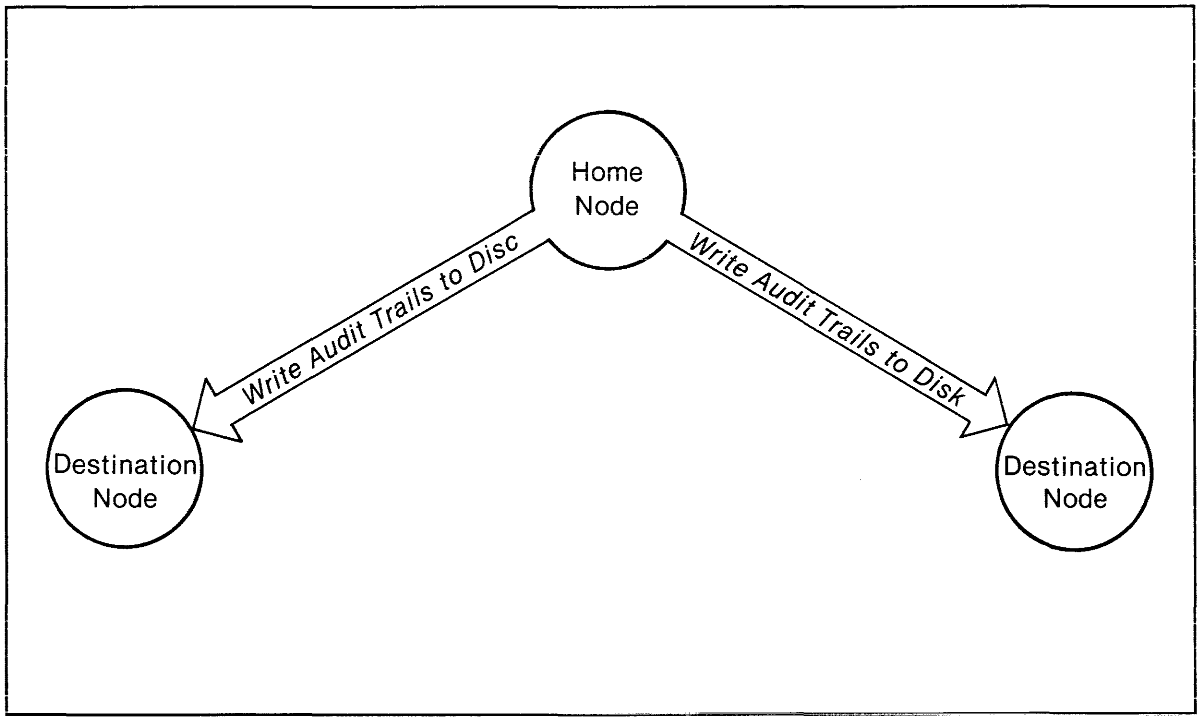


Figure 3-4. Phase 1 of Two Phase Commit

If any of the other nodes cannot be reached or reply that they cannot commit the transaction, transaction commit fails and the transaction is eventually backed out on all nodes. If the participating nodes all reply affirmatively, TMF writes a "transaction commit" record in the home node's audit trail and the transaction will then be recovered in the event of a total system failure. Successful completion of phase 1 requires communication between all nodes participating in a network distributed transaction. Figure 3-5 illustrates completion or failure of phase 1.

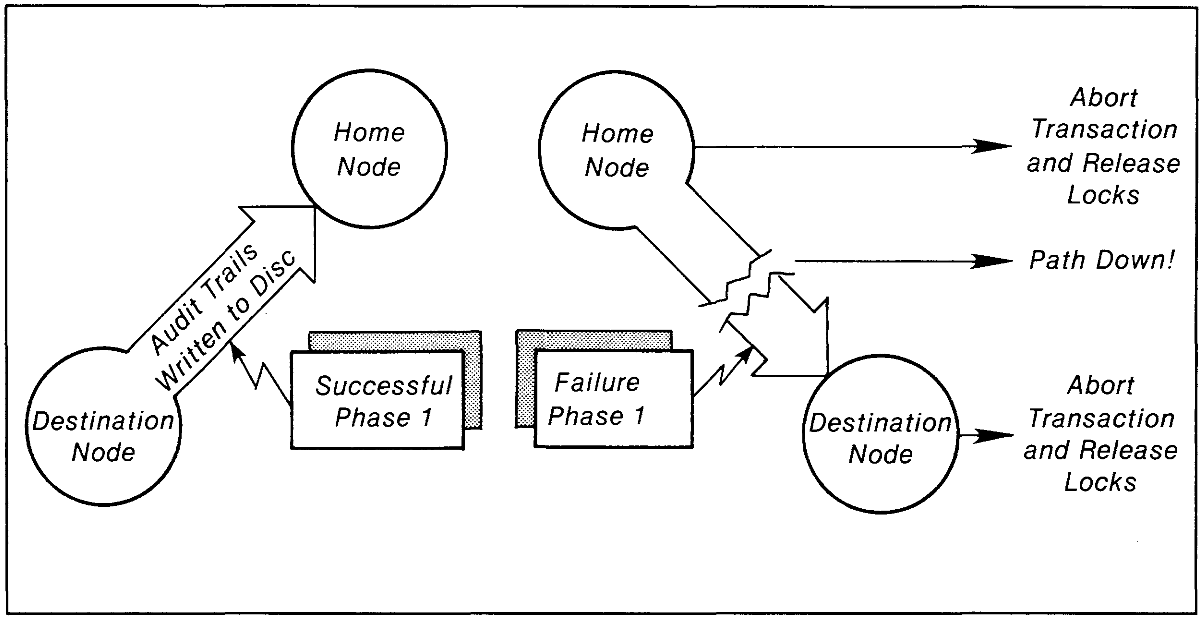

Home Node
Destination Node
Audit Trails Written to Disc
Successful Phase 1
Failure Phase 1
Home Node
Abort Transaction and Release Locks
Path Down!
Destination Node
Abort Transaction and Release Locks

Figure 3-5. Phase 1 of Two-Phase Commit—Completion or Failure

During phase 2, the Terminal Monitor Process on the home node sends a message instructing the destination nodes to release all locks held for the TRANSID . Once this message is sent and received, no further communication between nodes is necessary; locks are released independently at each node (Figure 3-6). If a participating node cannot be reached during phase 2, locks are released on all reachable nodes. When communication is restored to the unreachable node, the locks will be released and the records will be made available to other transactions.

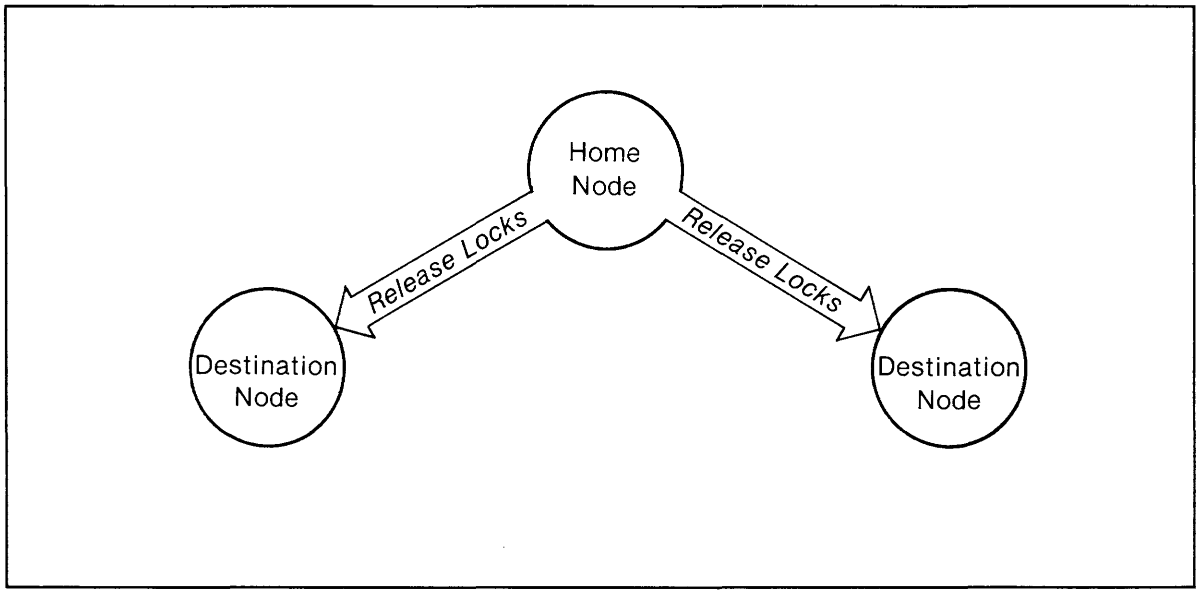


Figure 3-6. Phase 2 of Two-Phase Commit

SECTION 4

TMF DESCRIPTION

TMF solves several problems involved in maintaining the consistency of a database including:

- Maintaining images of the database changes (audit trails) caused by a transaction. If the transaction is network distributed, the audit trails are maintained on each node where the changes occur.

- Using the audit trails to back out failed transactions.

- Allowing selected portions of a database to be dumped to tape while normal processing continues against the database.

- Recovering a database or some portion of a database affected by a total system failure to a consistent state.

To solve these problems, several features—Audit Trails, Transaction Backout, Online Dump, and Rollforward—are provided by TMF. Each of these features is implemented by TMF system processes and a system with TMF can be viewed as having three interrelated aspects: (1) database and audit files, (2) TMF system processes, and (3) dump files. The rest of this section describes the TMF features and their relationship to the TMF proceses that implement them. The purpose of this section is provide enough detail about the processes to enable you to understand how to configure and control them through use of System Generation and the TMFCOM utility.

AUDIT TRAILS

An audit trail is a group of files that TMF uses to record information about a transaction's status and its changes to a database. Every transaction operation on a database file that the user designates as an "audited database file" is recorded on an audit trail. In this context, an "audited database file" is a file that is explicitly designated by the user as audited.

Audit trails have two parts:

1. A monitor audit trail containing transaction status information.

2. Auditprocess audit trails containing before and after images of records in the audited database files and transaction status information.

Monitor Audit Trails

One monitor audit trail has to be configured for every node in a network. The monitor audit trail contains timestamped records about the following events:

- Commit of a TRANSID on its home node.
- Abort of a TRANSID on its home node.
- The receipt of a distributed TRANSID from its source node; in this case the source node is recorded.
- Transmittal of a TRANSID to a remote node; in this case the destination node is recorded.
- Completion of phase 1 of the two-phase commit for a transaction that is outside its home node.
- The release of locks for a TRANSID on a destination node.
- Abort of a transaction on a destination node.

To configure the Monitor Audit trail, you must specify a primary file sequence and optionally an alternate file sequence. The alternate file sequence is a copy of the primary sequence. A sequence is a named group of files that is created by TMF as it builds the Monitor Audit trail. The sequence of files used for the monitor audit trail is specified by using the TMFCOM utility. TMFCOM allows you to designate the file extent sizes and generic file name of the primary and optional alternate monitor audit file sequence. The generic filename is specified as volume.subvolume.xx; for example $AUDIT.MONITOR.AA. The generic filename is used by TMF to create the files used in the monitor audit trail by appending, as required, a 6-digit sequence number to the generic filename each time a new file in the sequence is created; for example $AUDIT.MONITOR.AA000001, $AUDIT.MONITOR.AA000002. New files are created as old ones are filled.

A TMF system process called the Transaction Monitor Process (TMP) creates the files in the Monitor Audit Trail. The TMP is a system process pair defined by the user during system generation as a pseudo-logical device. One TMP has to be defined for each system using TMF on a network. Basically, it performs the following functions:

- Records the source node for transactions coming into the system.
- Records the destination nodes for transactions leaving the system.
- Writes the TRANSID abort record in the Monitor Audit Trail.
- Participates in the initial transmittal of a network distributed TRANSID.
- Creates new files, as needed, in the audit trail sequence.
- Purges old files in the audit trail sequence.
- Controls transaction backout.

Another TMF system process called the TMFMONITOR process writes transaction commit records to the Monitor Audit trail. The TMFMONITOR is specified during system generation as a system process that runs at a reserved PIN in each processor.

Auditprocess Audit Trails

Auditprocess audit trails are created and purged by the TMP and written by TMF Auditprocesses; each Auditprocess writes blocks of before and after images to its Auditprocess audit trail.

You define Auditprocesses during system generation as logical devices and there must be one Auditprocess for each disc controller group that contains audited discs. A disc controller group comprises those I/O controllers whose ownership must switch together. Each defined Auditprocess writes the Auditprocess audit trail for all volumes belonging to its particular disc controller group.

The sequence of files used for the Auditprocess audit trail is specified in the same manner as the files for the Monitor Audit trail with one important difference—multiple Auditprocess audit trails can be defined per system.

Depending on performance and disc utilization requirements that you establish, several Auditprocesses can write to the same Auditprocess audit trail or each Auditprocess can write to its own Auditprocess audit trail (Figure 4-1). The relationship of the Auditprocesses to their audit trails is specified through the TMFCOM utility.

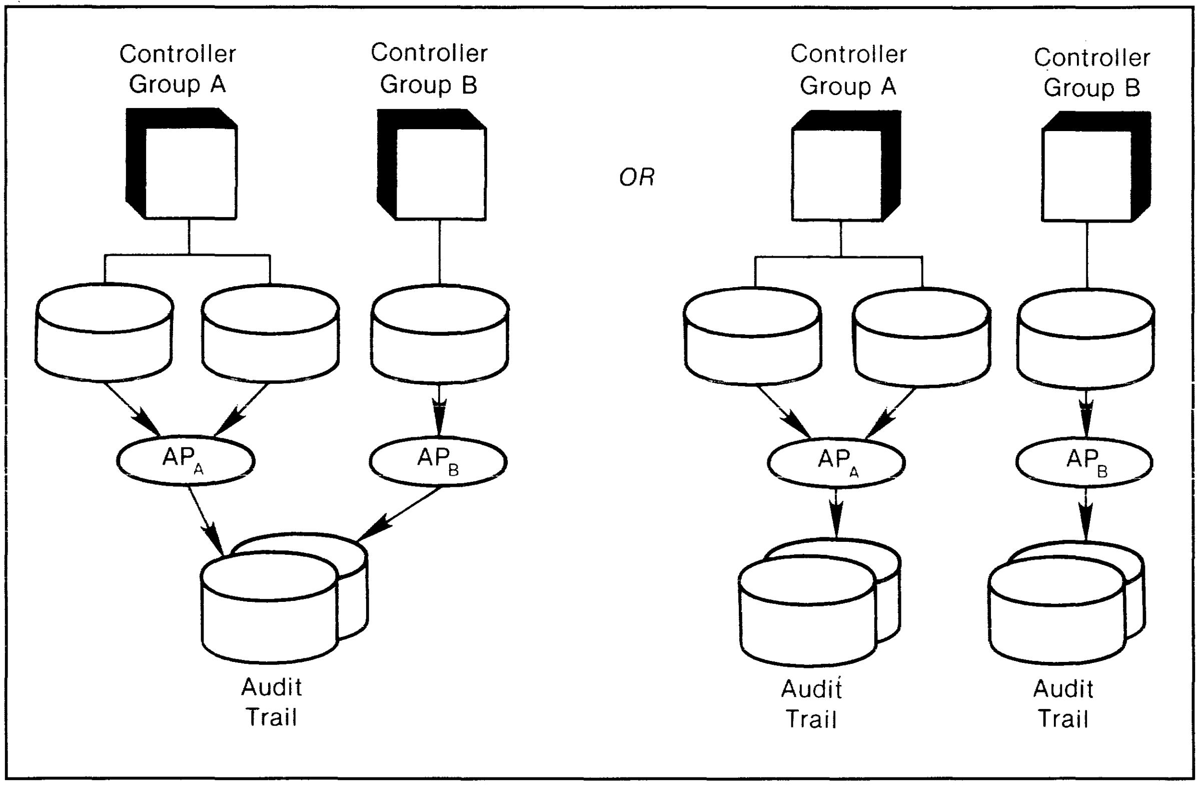


Figure 4-1. Auditprocesses and Audit Trails

Dumping Audit Trails

Audit trail are required for Rollforward recovery and transaction backout. Eventually, the amount of space required for audit trails could exceed the disc space available for them. Because changes to audited files cannot be made if there is no space for their audit trails, it is necessary to ensure that the audit trails are dumped periodically before the disc files fill.

Dumping involves copying some of the files in the audit trail sequence to tape—which will result in the space they occupy being freed—and saving the tapes for possible use in Rollforward recovery. If a file in the sequence contains before and after images for an active transaction (one that has not committed or aborted), it cannot be copied to tape. A TMF process called Auditdump is involved in dumping the audit trails. Auditdump can be configured to function automatically as audit files are filled or it can be run manually, by the operator, using the TMFCOM DUMP AUDITTRAIL command. Duplicate copies of the audit trail dump tapes can be made simultaneously.

TRANSACTION BACKOUT

Transaction Backout uses the before images in the audit trails to undo the effects of an aborted transaction. TMF's ability to back out transactions, without affecting database consistency, depends on three factors: (1) the concurrency control mechanism that prevents transaction interaction, (2) the two-phase commit mechanism, and (3) audit trails. These mechanisms were discussed, in detail, in previous parts of this publication. This section describes how backout is initiated and the TMF system processes involved in transaction backout.

Initiating Transaction Backout

A transaction is backed out when it aborts. Specifically, this means that before the transaction commit record was written to the Monitor Audit trail, something happened—either because of a decision by the application program, a component failure, or operator intervention—that stopped the transaction from committing. Since the effect of backout is the same as if the transaction had never started, an application program can recover by restarting the transaction from the beginning: a contrast to the normal NonStop technique of having the backup process take over from the point of failure and complete processing of the transaction.

A transaction is backed out for the following reasons:

- A PATHWAY Screen Cobol program issues the RESTART-TRANSACTION or ABORT-TRANSACTION verbs. RESTART-TRANSACTION starts the transaction from the BEGIN-TRANSACTION verb and ABORT-TRANSACTION terminates the transaction without restarting it. This is a voluntary decision on the part of the application and TMF provides equivalent mechanisms for non-PATHWAY users.

- The PATHWAY Screen Cobol program is suspended or aborted due to errors or specific PATHCOM commands.

- The cpu of a PATHWAY server that is doing work for a transaction fails. The transaction will be aborted and the TCP will restart the transaction automatically.

- The TCP primary process that started the transaction is deleted. The transaction's change will be aborted and the backup TCP process will restart the transaction. If there is no backup process, the transaction will be aborted but not restarted.

- The user enters a TMFCOM command to abort the transaction.

- Network communication is lost between participating nodes of a network distributed transaction before the transaction is committed. This situation is described in more detail in the following subsection.

What Happens During Transaction Backout

During transaction backout, a TMF process called the Backout Process writes the before images, saved on the Auditprocess audit trails, back to the files that were affected by the failed transaction. After the files are restored, the locks for the TRANSID will be released.

Backout For Network Distributed Transactions

For a network distributed transaction, loss of communication between participating nodes can result in transaction backout. The situations where this can occur are:

1. A loss of communication between destination and source nodes occurs before the transaction has committed. The nodes affected by the communications loss will each abort and back out the transaction (releasing its locks) and it will eventually be aborted and backed out on all other nodes involved in the transaction (Figure 4-2).

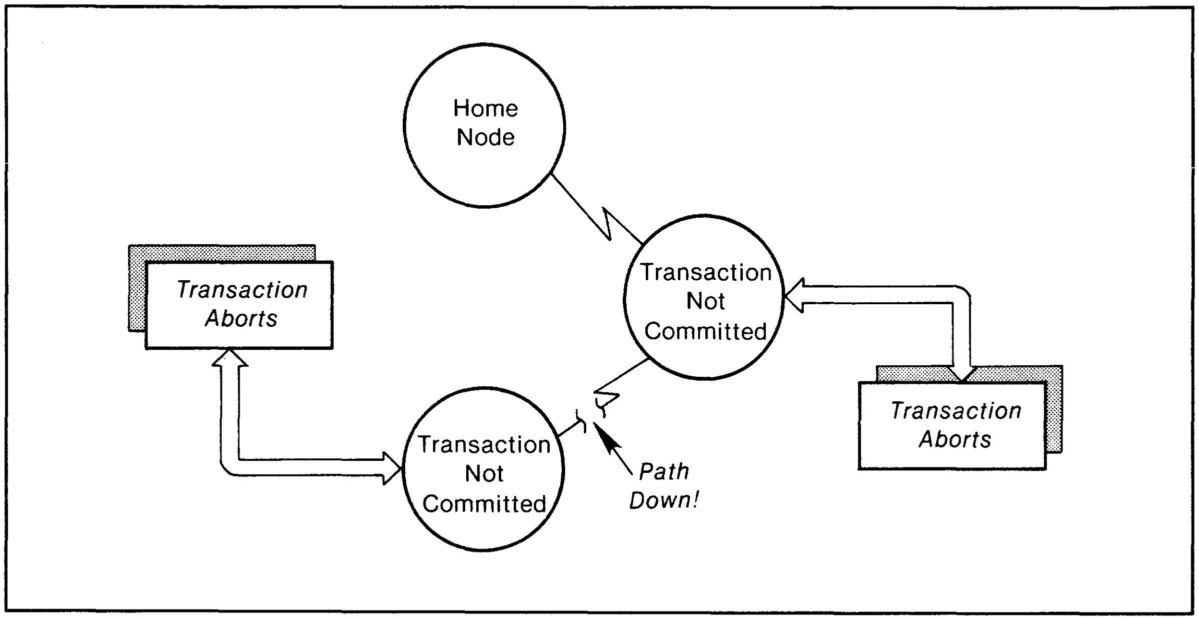


Figure 4-2. Network Backout Before Changes are Committed

2. END-TRANSACTION has been issued on the home node and phase 1 of the Two-Phase Commit fails either because of a path down to a participating node or a participating node replying that it cannot commit the transaction. The transaction will be aborted and backed out on all nodes participating in the transaction (Figure 4-3).

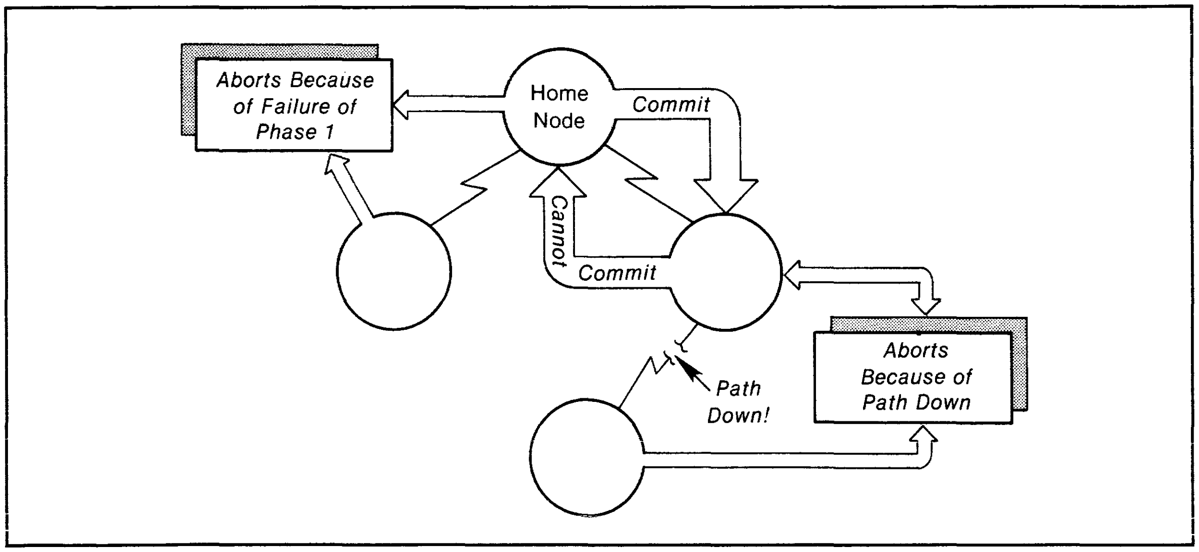

Aborts Because of Failure of Phase 1
Home Node
Commit
Cannot Commit
Aborts Because of Path Down
Path Down!

Figure 4-3. Network Backout During Phase 1 of Two Phase Commit

ONLINE DUMP

The Online Dump feature allows users to copy audited database files to tape. Information about each copied file is maintained in the TMF catalog. The function of Online Dump is to preserve copies of audited database files as protection against total system failure; each online dump of a file provides an image of the file that can be used by TMF's Rollforward feature to reconstruct the file.

Online Dump can be used to copy files that are open and changing; it is not necesssary to stop transaction processing against a file while copying it.

Online Dump is an operation that involves several components:

• The TMF catalog	contains the control information related to various dumps to tape.
• The TMF Catalog Process	a TMF process that provides access to the catalog.
• The TMFCOM DUMP FILES command	user command that starts Online Dump.

TMF Catalog

The TMF Catalog is a set of disc files. The files are on the same mirrored disc volume as the TMP control files and they contain two types of control information related to Online Dump:

1. A dump directory containing a history of the files that were dumped to tape and when they were copied. This includes:

    - The names of the tape drives used to dump the files.
    - A timestamp and unique serial number for the dump.
    - The generation number of the dump. Since there can be different dumps of the same file, the generation number identifies each distinct version.
    - The copy number of the dump. Multiple copies can be made for each generation and the copy number identifies each distinct copy of a particular version.
    - The generic name and sequence number of the Auditprocess audit trails in use at the time of the Online Dump.
    - The sequence number of the Monitor audit trail in use.
    - Serial numbers that associate a tape reel with a particular dump.
    - Additional information related to dumping audit trails and the TMF Catalog itself onto tape.

2. A tape directory that contains all known tape reel identifiers and their current status: scratch, known bad, or assigned to a particular dump serial number.

TMF's Rollforward recovery uses information in the catalog to determine the tapes needed for its operations.

TMF Catalog Process

The TMF Catalog process is a NonStop server process, executed during backup, that writes the information, related to Online Dump, to the TMF Catalog in a failsafe manner. One TMF Catalog process must be configured, during system generation, for each system that uses TMF.

TMFCOM DUMP FILES Command

You start Online Dump by issuing the DUMP FILES command and indicating which audited files are to be dumped. After the command is issued, the following steps take place:

1. The generic name and sequence number of the Auditprocess audit trail associated with the specified audited files are entered into the TMF catalog.

2. The sequence number of the current Monitor audit trail file is entered into the TMF catalog.

3. An online dump mark is written to the Auditprocess audit trail for each audited file opened by online dump. Because Online Dump writes to the Auditprocess audit trails, they must be configured (as discussed previously in this section) before the audited files can be backed up.

4. A serial number in the catalog, for the dump, is associated with a specific tape label.

5. The operator is prompted to mount a specific tape reel on a specific tape drive.

6. The operator either mounts the specified reel or replies that he is mounting another reel and specifies its identifier.

7. The label information for the reel is written to the TMF catalog.

8. The audited files are copied to the tape with labels describing the file.

9. A marker is written to the TMF catalog indicating that the dump was successful. Without this indication, the assumption is that the information on the tape is bad and will not be used during Rollforward.

Tape Labelling for TMF

Because Rollforward recovery of the database depends on the data contained on the Online Dump and Auditdump tapes, TMF has extended GUARDIAN tape handling by supporting labelled tapes and allowing you to specify an operator interaction unit.

The labelled tapes are defined according to the ANSI X3.27-1978 "American National Standards Magnetic Tape Labels and File Structure for Information Interchange". Using the labelled tapes protects the data on the tape from overwriting, thus ensuring the availability of their contents for Rollforward recovery. Only audited files can be copied to labelled tapes.

The operator interaction unit enables you specify a central location for tape control, consolidating the function by making it the responsibility of the operator. All requests for tape mounting will be sent to this unit.

ROLLFORWARD

Rollforward is used after:

- A total system failure ... to recover a database to its most recent consistent state prior to the failure.
- An audited file or volume is affected by a disc media failure ... to recover the data on the affected files.

Rollforward involves the following steps:

1. There has been a total system failure and the operator issues the TMFCOM command START TMF.
2. The TMFCOM utility replies to the TMFCOM command by informing the operator that there has been a failure; no audited files affected by the failure can be opened until they have been recovered.
3. The user issues the TMFCOM command RECOVER FILES (optionally specifying a subset of the files to be recovered).
4. TMF prompts the operator to mount the tape reels required for recovery of the files. Once the tape reels are mounted, Rollforward uses the Online Dump tapes and the audit trails to recover the files to a consistent state. The database files are first restored from the Online Dump tapes. After the files are restored, transactions that committed before the failure will be included in the recovery by having their after images written to the restored files.
5. As each file is recovered, a message indicating its status is displayed and that file can then be opened. The results of all transactions that changed the file and committed prior to the failure will be in the recovered file. This includes transactions that completed and sent a positive response to the user and transactions that committed but the response was not received before the failure.

SECTION 5

USING TMF — CONSIDERATIONS

Detailed information for the topics in this section can be found in the TMF Reference Manual that will be available with the release of TMF. The information in this section is intended for planning purposes only and covers:

- System management responsibilities and guidelines related to configuring and operating TMF in a system.
- Application programming rules and guidelines for systems with TMF.

SYSTEM MANAGEMENT RESPONSIBILITIES

System management involves:

- Using SYSGEN to configure TMF for your installation.
- Using the TMFCOM utility to configure TMF.
- Using the TMFCOM utility to operate TMF.
- Using FUP to designate audited database files.

Using SYSGEN To Configure TMF

The steps involved in using SYSGEN for TMF are illustrated in Figure 5-1.

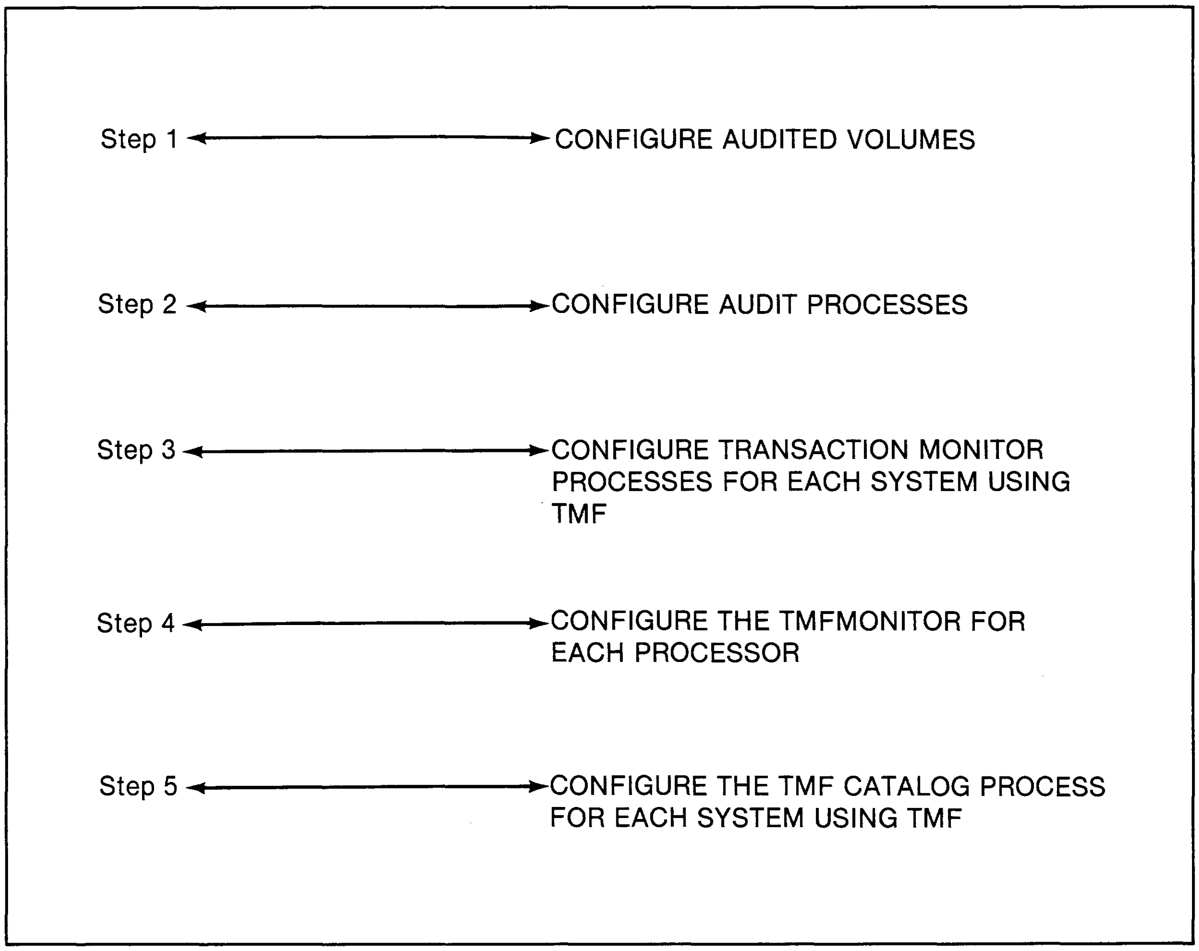

Step 1
CONFIGURE AUDITED VOLUMES
Step 2
CONFIGURE AUDIT PROCESSES
Step 3
CONFIGURE TRANSACTION MONITOR PROCESSES FOR EACH SYSTEM USING TMF
Step 4
CONFIGURE THE TMFMONITOR FOR EACH PROCESSOR
Step 5
CONFIGURE THE TMF CATALOG PROCESS FOR EACH SYSTEM USING TMF

Figure 5-1. TMF System Generation

STEP 1. Decide which disc volumes will contain audited database files. Once you've made this decision, specify the number of pages of physical memory to be allocated for the audit buffer pools in the I/O configuration entry for each disc. Before and after images of all changes to the database files are written, by the Discprocesses for the audited volumes, to the audit buffers (Figure 5-2). Phase 1 of the Two Phase Commit forces the records in the audit buffers to be written to the audit trail. The size of the buffer pools is specified using the parameter audit buffer pool size: a new parameter in the disc configuration entry. The disc can contain audited database files only if this parameter is nonzero.

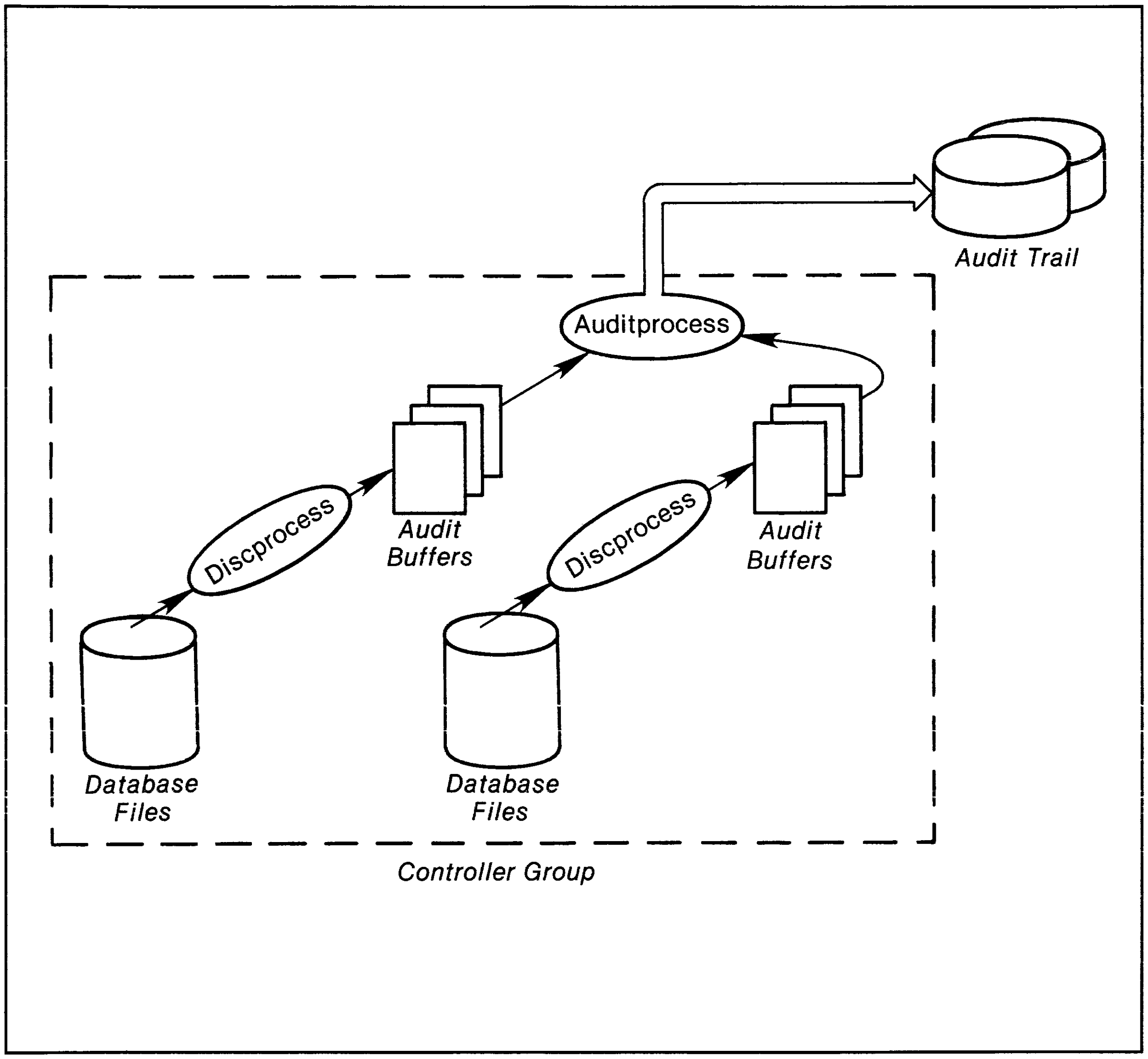


Figure 5-2. Audit Buffers

STEP 2. Configure Auditprocesses as logical devices. This involves:

- Using the standard SYSGEN I/O device format to specify one Auditprocess for each disc controller group that contains audited volumes. The Auditprocess will write the Audit trails for the files on the volumes belonging to that controller group. The Using TMFCOM to Configure TMF part of this section describes how to specify audit trails and relate them to the Auditprocesses configured during system generation.

- Declaring the Auditprocess in the system process entries in the processor section of the primary and backup cpus.

- Declaring a controller group that contains the Auditprocess controller and the primary controller of the audited disc volumes. The Auditprocess controller is a dummy controller; its only purpose is to associate the Auditprocess with the Discprocess of the audit volumes.

Each Auditprocess that you configure removes one word-addressable page from system data space. Additionally, a number of pages equivalent to the audit buffer pool size per audited disc are removed from physical memory.

The TMFCOM command ADD AUDITTRAIL is used to relate an Auditprocess to a specific Audit trail. This command is discussed in the next part of this publication.

STEP 3. Use the standard SYSGEN I/O device format to configure one Transaction Monitor Process (TMP) for each system that uses TMF and add its process number to the processor information section of the primary and backup cpus.

The configuration parameter, specified in the device entry, will be the logical device number of the disc volume where the TMP keeps its control information.

STEP 4. Declare the TMFMONITOR in the system processor entries for each processor in your system.

STEP 5. Use the standard SYSGEN I/O device format to configure one TMF Catalog Process for each system that uses TMF and add its process number to the processor section of the primary and backup cpus.

Using the TMFCOM Utility to Configure TMF

You use configuration commands to create, alter, and determine the status of objects such as Audit trails or Auditdump processes that TMF requires or uses in its operations. Table 5-1 summarizes the functions of the TMFCOM configuration commands.

Table 5-1. TMFCOM Configuration Commands

Command	Function
ADD AUDITTRAIL	Create Audit trails and relate them to SYSGEN-configured Auditprocesses.
INFO AUDITTRAIL	Display configuration and status information for Audittrails.
ADD AUDITDUMP	Specify Auditdump process(es) that will be created when TMF is started. Auditdump processes will automatically dump Audit trails to tape as they fill. If Auditdump is not configured with this command, it must be run manually.
DELETE AUDITDUMP	Delete Auditdump process(es).
INFO AUDITDUMP	Display status information for Auditdump process(es).
ALTER AUDITDUMP	Allows operator to specify tape drives and reels to be used for Auditdump.
ALTER BACKOUT	Modify the cpu location and priority of the Backout process.
INFO BACKOUT	Display the current cpu location and priority of the Backout process.
ALTER TMF	Specify the terminal or process to use as operator interaction unit. All TMF requests for tape mounting assistance will be sent to this unit.
INITIALIZE TMF	Purge the TMF configuration files prior to reinitializing TMF.
INFO TMF	Provides a complete listing of the current audit subsystem configuration.
INITIALIZE CATALOG	Configure attributes of TMF catalog.
ALTER CATALOG	Modify attributes of TMF catalog.
INFO CATALOG	Display current configuration of TMF catalog.

CONFIGURATION CONSIDERATIONS. The following are some preliminary guidelines related to configuring TMF.

- ADD AUDITTRAIL allows you to define the set of files that form an audit trail and specify the Auditprocess(es) that write to the audit trail. The files are identified by their generic filename—volume.subvolume.xx. The volume specified in the generic filename must be a mirrored volume. It is recommended that this volume: (1) should not belong to the same controller group as the Auditprocess(es) that write to it and (2) should not contain any audited database files. That is, audited database files should be on a different disc from their audit trails.

- ADD AUDITTRAIL allows you to define the the number of extents occupied by each file in the sequence. The total amount of space you require on the audit trail disc is dependent on the transaction update rate which determines: how quickly the audit disc will fill and how often Auditdump needs to be run (either automatically or manually). As a rough approximation, you can use the following formula to determine how the transaction update rate affects the amount of audit trail disc space.

$$\left(\begin{array}{r} 1.3 \text{ x number of bytes inserted/hour} \\ + \ 1.3 \text{ x number of bytes deleted/hour} \\ + \ 2.3 \text{ x number of bytes modified/hour} \end{array}\right) = \begin{array}{l} \text{number of bytes} \\ \text{written to audit} \\ \text{files per hour} \end{array}$$

$$\begin{array}{lll} & \text{inserted/hour} & = \text{number of records inserted per hour} \times \text{size of records} \\ \text{where} & \text{deleted/hour} & = \text{number of records deleted per hour} \times \text{size of records} \\ & \text{modified/hour} & = \text{number of records modified per hour} \times \text{size of records} \end{array}$$

For example, 60 transactions do a total of 1000 inserts,1000 deletes, and 1000 modifications per hour to a file containing 50 byte records. The number of bytes written to the audit trail in one hour will be approximately $(1.3 \times (2 \times 50000)) + (2.3 \times 50000) = 245{,}0000$ bytes.

If you plan to dump the audit trail disc every four hours, a total of 980,000 bytes or 478 pages would be required on the disc. Assuming that the audit trail sequence is spread over 6 files, each file can be configured to occupy 79 (478/6) pages or extent sizes of 5 primary (79/16) and 5 secondary.

Using the TMFCOM Utility to Operate TMF

You use TMFCOM operational commands to control and run TMF after it has been configured. There are four types of operational commands:

1. Controlling TMF.
2. Controlling the TMF catalog.
3. Manually Controlling Transactions.
4. Dumping and recovering the database.

Table 5-2 summarizes the TMFCOM operational commands.

Table 5-2. TMFCOM Operational Commands (Part 1 of 2)

Command	Function
	Controlling TMF—Commands
START TMF	Start TMF before transactions can write or lock audited database files.
STOP TMF	Stop TMF on a system without active transactions (see note).
STATUS TMF	Indicates the current state of the TMF Audit subsystem.
STOP CATALOG	Prevent access to TMF catalog (see note).
START CATALOG	Used only after STOP catalog to start automatic operation of Online dumps and AUDITDUMPS.

Note: The database can be recovered (by Rollforward) to a point in time immediately after a STOP TMF command if there are archived copies of the Auditdumps and Online Dumps available. This may be necessary if the online database and the online audit media have been destroyed and are unavailable for normal Rollforward recovery. To recover to a STOP TMF point, first issue STOP CATALOG and then copy the following to tape:

- The Monitor Audit trail.
- The Auditprocess Audit trails.

After stopping TMF, dump a copy of the TMF Catalog using the DUMP CATALOG command. If the preceding tapes and the online dump tapes have been stored in a safe place and are available, Rollforward can use them to recover the database.

Command	Function
	Controlling TMF—Commands
STATUS CATALOG	Display the status of the TMF catalog.
STATUS AUDITTRAIL	Display the current status of the Audit trail.
NEXT AUDITTRAIL	Close the current Audit trail file and open the next one in the sequence.
STATUS AUDITDUMP	Display the status of an Auditdump process.
CONTROL AUDITDUMP	Modify the status of an Auditdump process.
	Controlling the TMF Catalog—Commands
ADD DUMPS	Add a dump(s) to the dump directory of the TMF catalog.
INFO DUMPS	Displays dumps recorded in the TMF Catalog.

Table 5-2. TMFCOM Operational Commands (Part 2 of 2)

DELETE DUMPS	Purges tape files or file sets from the dump directory of the TMF Catalog.
ALTER DUMPS	Modify the TMF catalog entry for a dump.
ADD TAPES	Add tape reel(s) to tape directory of TMF catalog.
INFO TAPES	Displays reels of tape recorded in the tape directory of the TMF Catalog.
DELETE TAPES	Deletes tape reel(s) from the tape directory of the TMF catalog.
ALTER TAPES	Change current status of a tape record in tape directory of TMF catalog.
RECOVER CATALOG	Restore TMF catalog from a tape dump produced by using DUMP CATALOG.
DUMP CATALOG	Dump the TMF catalog to a tape volume and record the dump in the catalog.
	Dumping and Recovering the Database—Commands
DUMP FILES	Starts Online Dump.
DUMP AUDITTRAIL	Dumps specified audit trails from disc totape.
RECOVER FILES	Starts Rollforward.
	Manual Transaction Control Commands
END TRANSACTION	Forces a committed transaction's records to be unlocked.
ABORT TRANSACTION	Backs out an uncommitted transaction andunlocks its records.
STATUS TRANSACTION	Display the TRANSID and status oftransactions.

Using FUP to Designate Audited Database Files

Your SYSGEN configuration specified the disc volumes that could contain audited database files. Specific files on the volumes can be designated as audited when they are first created using FUP CREATE or by using FUP ALTER for existing files. A [NO]AUDIT parameter has been added to CREATE and ALTER to support TMF. FUP ALTER can also be used to change the status of a file from audited to non-audited.

APPLICATION PROGRAMMING

This section:

- Summarizes the functions of the procedures and verbs provided to allow Screen Cobol, TAL, COBOL, and FORTRAN programmers to use TMF.
- Describes the record locking rules that programmers follow to write server processes.
- Discusses how to avoid deadlock.
- Presents some preliminary guidelines for designing transactions in a PATHWAY environment.

Screen Cobol

The Screen Cobol verbs that enable PATHWAY applications to use TMF are:

- BEGIN-TRANSACTION
- END-TRANSACTION
- ABORT-TRANSACTION
- RESTART-TRANSACTION

BEGIN-TRANSACTION ... identifies the beginning of a sequence of operations that will be treated as a single transaction. When this verb is executed: the terminal enters transaction mode, TMF starts a new transaction, and the new TRANSID is assigned to the terminal. If the transaction is aborted for any reason—with the exception of the Screen Cobol program issuing ABORT-TRANSACTION—its changes will be backed out by TMF and restarted at the BEGIN-TRANSACTION point with a new TRANSID.

ABORT-TRANSACTION ... is used to abort the transaction. TMF will back out all database modifications made by the transaction and the transaction will not be restarted.

END-TRANSACTION ... identifies the end of the transaction and indicates that the transaction's database changes should be committed. After END-TRANSACTION is executed, the transaction cannot undo any of its changes and the terminal is switched out of transaction mode.

RESTART-TRANSACTION ... is used to indicate the transaction has failed and that it should be backed out and restarted at the BEGIN-TRANSACTION point with the same TRANSID.

TAL Procedures

TAL programmers use TMF by calling the following procedures:

- BEGINTRANSACTION
- ENDTRANSACTION
- ABORTTRANSACTION
- RESUMETRANSACTION
- ACTIVATERECEIVETRANSID
- GETTRANSID

BEGINTRANSACTION ... causes a new TRANSID to be created by TMF. The new TRANSID becomes the current process TRANSID for the process that called BEGINTRANSACTION. An optional reference parameter tag can be specified with this call. TMF will return the tag of the new TRANSID to this parameter.

ENDTRANSACTION ... causes the changes associated with the current process TRANSID to be committed.

ABORTTRANSACTION ... causes the changes associated with the current process TRANSID to be aborted and backed out. The programmer can restart the transaction at the BEGINTRANSACTION call under a new TRANSID.

RESUMETRANSACTION ... is used by programmers coding multi-threaded requester processes (like the TCP). This procedure is called with the transaction tag returned from the BEGINTRANSACTION call. The TRANSID identified by the tag becomes the current process TRANSID for the process calling RESUMETRANSACTION.

ACTIVATERECEIVETRANSID ... is used by programmers coding $RECEIVE queuing servers: that is, servers that queue messages from several requesters concurrently before replying (Figure 5-3).

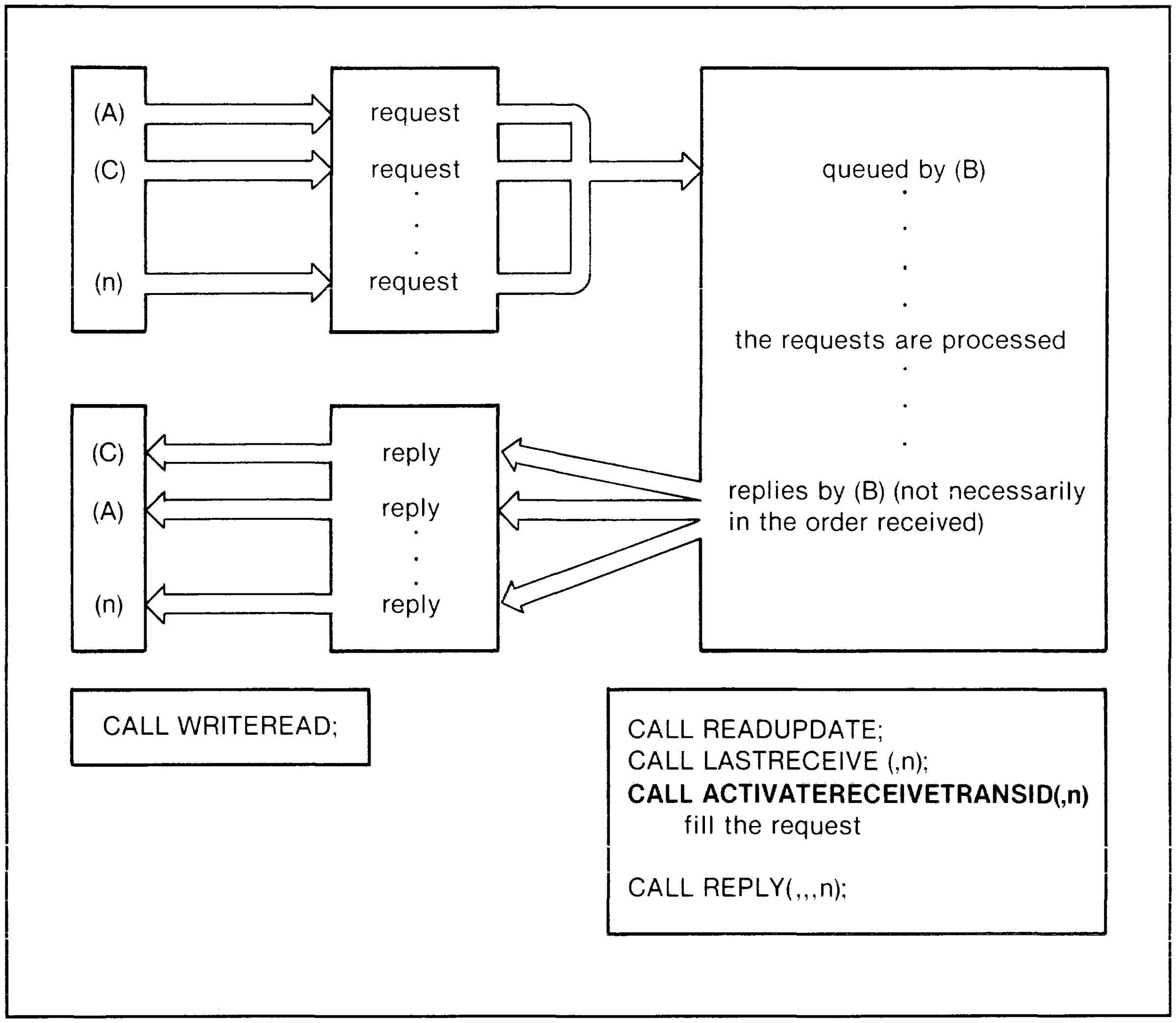


Figure 5-3. $RECEIVE Queuing Servers

The server identifies the requester associated with the message by obtaining its message tag through a call to the GUARDIAN LASTRECEIVE procedure and indicates which message it is responding to by including the message tag when it replies to the message. Since $RECEIVE queuing servers could be doing database operations for several requesters concurrently, they need to acquire the current process TRANSID dynamically. That is, whenever they do some operations for a request message, they need to assume its TRANSID for the duration of the operations and then acquire the TRANSID of the next message. A call to ACTIVATERECEIVETRANSID after a call to LASTRECEIVE allows the server to specify that the TRANSID of the message identified by message tag should be current for the process.

GETTRANSID ... is used to obtain the current process TRANSID of the calling process.

COBOL and FORTRAN

COBOL and FORTRAN programmers can use the ENTER TAL verb and CALL statement to use the TAL TMF procedures. However, because of the single-threaded nature of most COBOL and FORTRAN I/O, it is not recommended that you write modules that have more than one transaction outstanding at any given time. That is, avoid writing multi-threaded requesters and servers.

Record Locking

The TMF Operations section described the record locking rules that TMF enforces to prevent transaction interaction. The following is a summary of the rules you follow to write a transaction.

1. Lock all records that are updated or deleted by the transaction. Locks can be acquired for an entire file (LOCKFILE) or on a record-by-record basis.

   For example, in COBOL:

   ```
     READ EMP-MASTER WITH LOCK KEY IS EMP-NO-KEY OF EMP-REC.
     IF EMP-FIELDS OF EMP-REC = SAVED-EMP-FIELDS OF EUPDATE-MSG PERFORM
     970-REWRITE-EMP

   970-REWRITE-EMP.
     REWRITE EMP-REC WITH UNLOCK.
   ```

   TMF will verify that the record is locked and reject the REWRITE if the record has not been locked. The locks will be held until the transaction either aborts and is backed out, or commits. If a transaction issues LOCKFILE, modifies records while it holds the lock granted by LOCKFILE, and then issues UNLOCKFILE prior to issuing END-TRANSACTION, the file lock will be held until the transaction commits. Alternatively, a transaction could issue UNLOCKFILE when it does not own the file lock but does own record locks that were acquired individually. In this case, only the locks for records not updated by the transaction will be released prior to transaction commit.

2. Lock records that are read and used by the transaction in producing its output, but not changed. Following this rule guarantees that the transaction's reads are repeatable.

   For example:

   ```
     READ EMP-MASTER WITH LOCK KEY IS EMP-NO-KEY OF EMP-REC.
     IF EMP-FIELDS OF EMP-REC = SAVED-EMP-FIELDS OF EUPDATE-MSG
        .
        .
     ELSE
       UNLOCKRECORD EMP-MASTER
      .
   ```

   In this example, the lock will be released if the record is not modified.

When Locks are Released—TMF and ENSCRIBE

Table 5-3 shows the difference between transaction mode record locking (locks identified by TRANSID) and record locking for ENSCRIBE (locks identified by processid and openid).

Table 5-3. Lock Release TMF and ENSCRIBE

	unmodified		**modified**	
	record	file	record	file
ENSCRIBE	at unlock or close	at unlock or close	at unlock or close	at unlock or close
TMF	at unlock or endtrans	at unlock or endtrans	at endtrans	at endtrans

Notes:

unlock	— When UNLOCKRECORD or UNLOCKFILE, as appropriate, is called.
close	— When closing the file in which the objects are locked.
endtrans	— When ENDTRANSACTON completes or an aborted transaction is backed out.

Avoiding Deadlock

The following is an example of a sequence of record locking operations that results in a deadlock situation:

- Transaction 1 locks record A.
- Transaction 2 locks record B.
- Transaction 1 attempts to lock record B and has to wait.
- Transaction 2 attempts to lock record A and has to wait.

Neither transaction can proceed and the situation is a deadlock.

There is no way of detecting if a transaction becomes involved in a deadlock. However, the following situations can be detected:

- A transaction is attempting to read or lock a record that is already locked.
- A transaction's read or lock request is waiting too long before completion.

Each of these situations is explained below and illustrated in Figure 5-4. If either of these situations occurs, you can assume (although it may not be true) that the transaction is in a deadlock and code it to abort. The locks held for the transaction will be released, avoiding the possibility of it participating in or prolonging a deadlock.

I/O REQUEST WAITING TOO LONG. In default locking mode, TAL programmers can determine if a request has waited too long before completion. In this mode, a process will be suspended when it attempts to access a locked record. To avoid deadlock, open the file using no-wait I/O and specify a time limit < > 0 in the call to AWAITIO. If AWAITIO returns GUARDIAN error 40 (indicating timeout), the transaction may be in a deadlock situation.

COBOL programmers can open files using the new WITH TIME LIMITS parameter. WITH TIME LIMITS indicates that further I/O requests will be timed by specifying a value in the TIME LIMITS parameter of the request. If the I/O request times out, GUARDIAN error 40 will be returned to the request.

FORTRAN programmers can open files with the TIMED specifier and use the TIMEOUT specifier in their I/O requests to specify a timeout value. If the I/O request times out, GUARDIAN error 40 will be returned to the request.

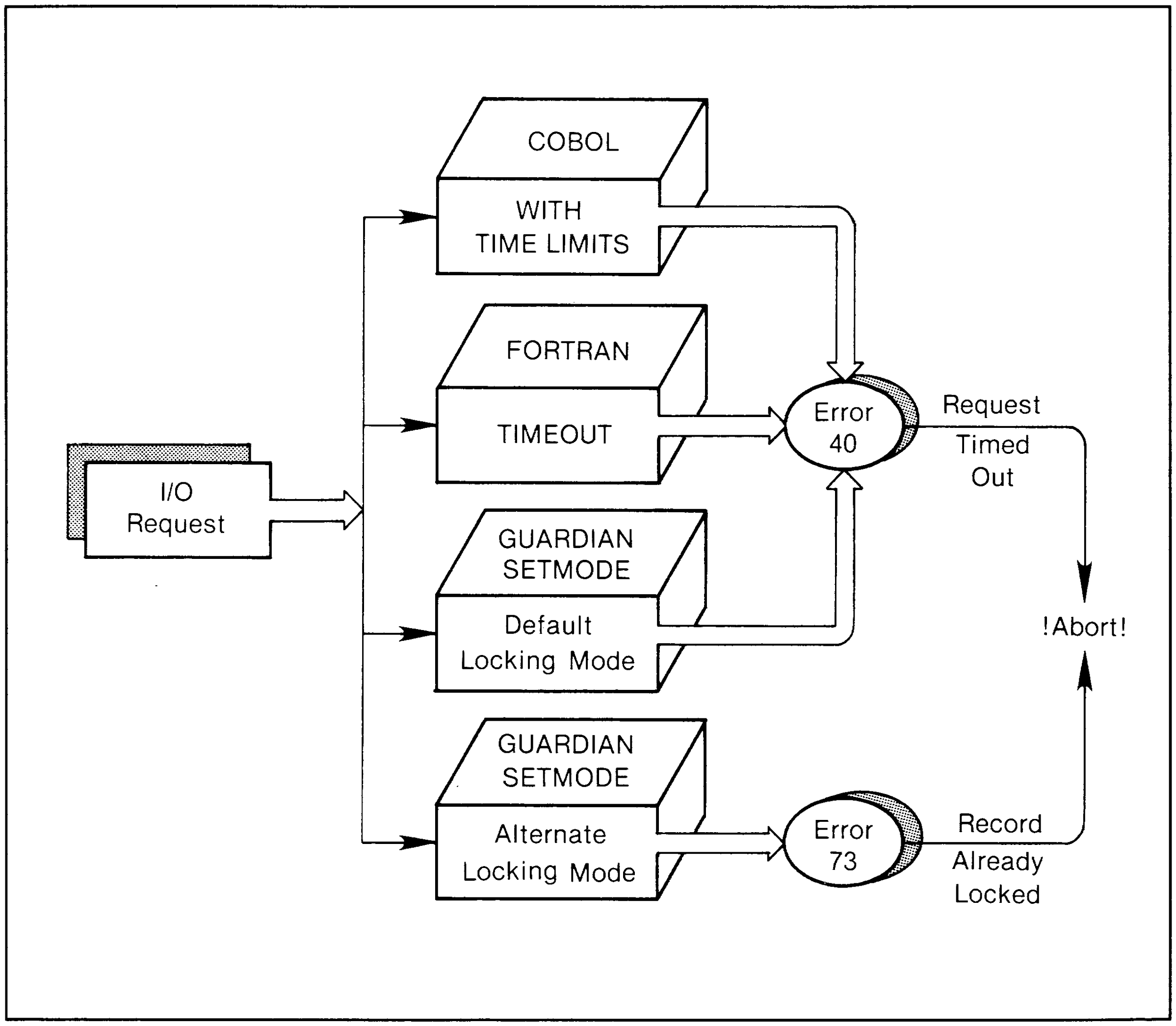


Figure 5-4. Avoiding Deadlock

RECORD ALREADY LOCKED. TAL programmers can determine if a record is already locked by using the GUARDIAN SETMODE procedure to select alternate locking mode. In this mode, file error 73 will be returned to the request when it attempts to access a locked record.

Transaction Design Guidelines For PATHWAY

This section presents some preliminary guidelines for designing PATHWAY applications that use TMF. Basically transaction design involves the following primary questions. What is the logical unit of work that you want to accomplish within an application (e.g. order entry or account balancing) and how can it be divided into a number of transactions that can be recovered by TMF? Factors that influence the answers to these questions are:

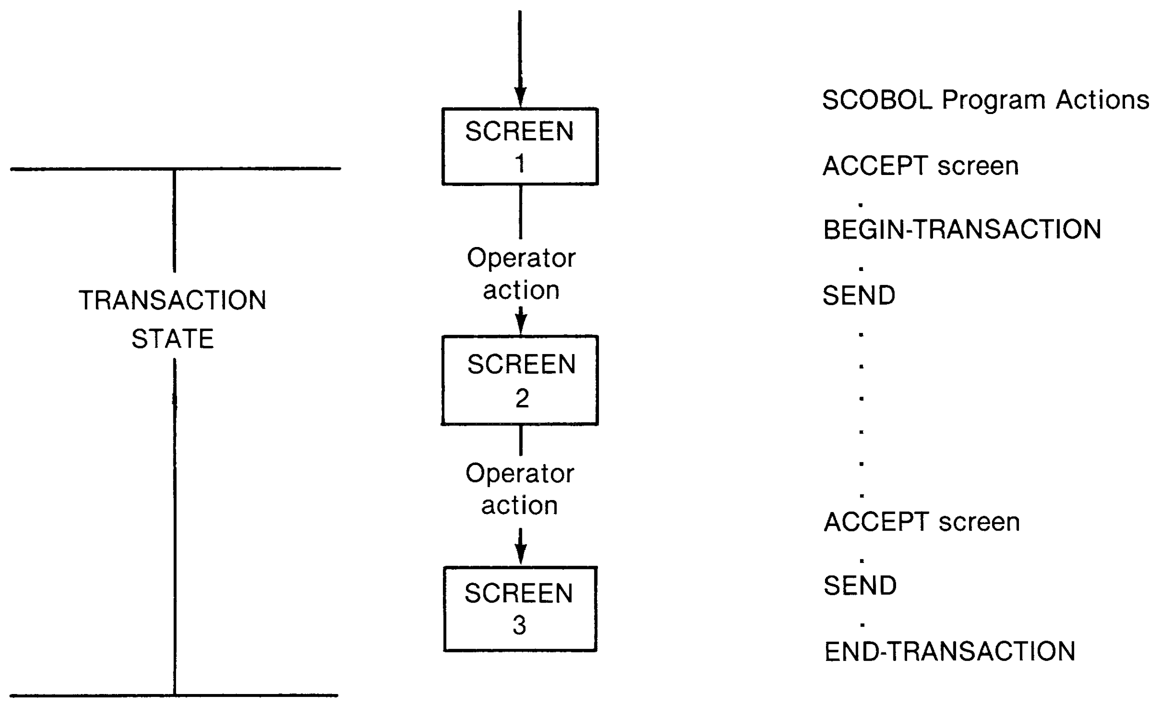
• Concurrency	how long will record locks be held for a transaction?
• Performance	how much extra disk I/O, server activity, and TCP paging is involved in the choice of one design over another?
• Consistency	how large is the unit of recovery for each transaction and is it adequate for the application?

Transactions can be divided into two categories.

1. Simple Single Screen Transactions ... all the data required by the transaction can be entered by the operator into one screen. This type of transaction is straightforward and the preceding questions do not apply.

2. Multi-Screen Transactions ... several screens are required to accumulate the data for the transaction. There are three possible types of multi-screen transactions: (1) long transactions, (2) context-saving transactions, and (3) multiple short transactions. Each of these types is described in the following pages along with some of their possible advantages and disadvantages.

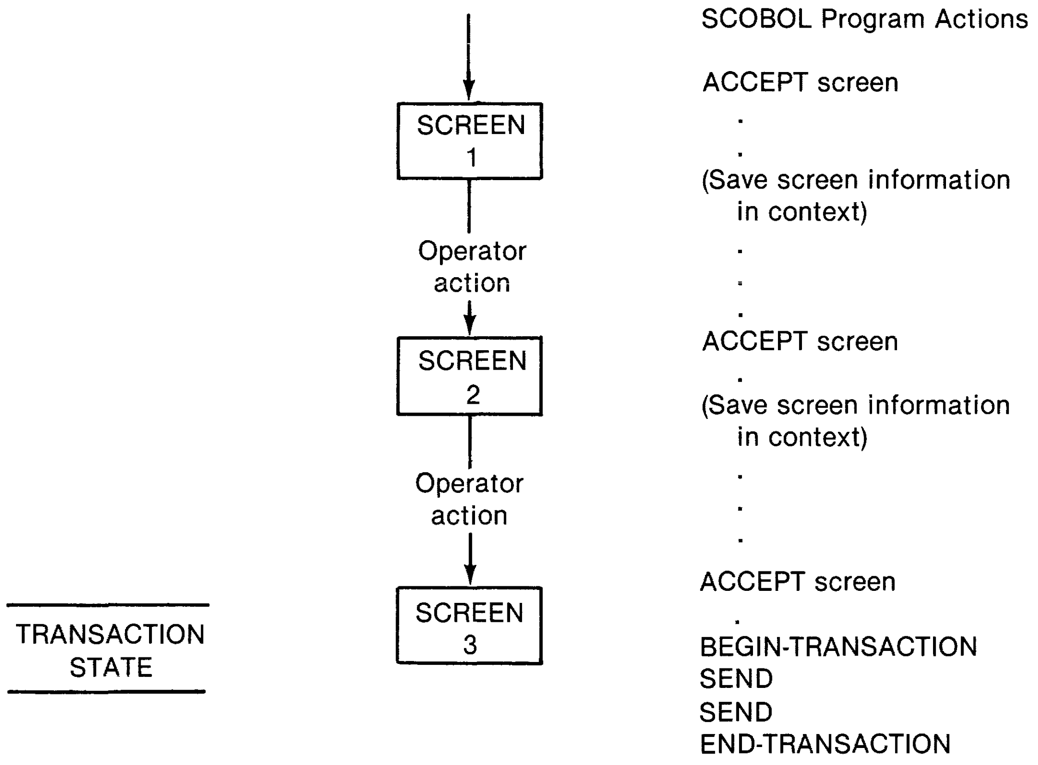
LONG TRANSACTIONS. A typical sequence of Screen Cobol actions for a long transaction is illustrated below.



Advantages: Failure at any point in transaction state will cause all updates to audited files to be backed out.

Disadvantages: Resources may be locked for an indeterminate amount of time.

CONTEXT-SAVING TRANSACTIONS. A typical sequence of Screen Cobol actions for context-saving transactions is illustrated below.

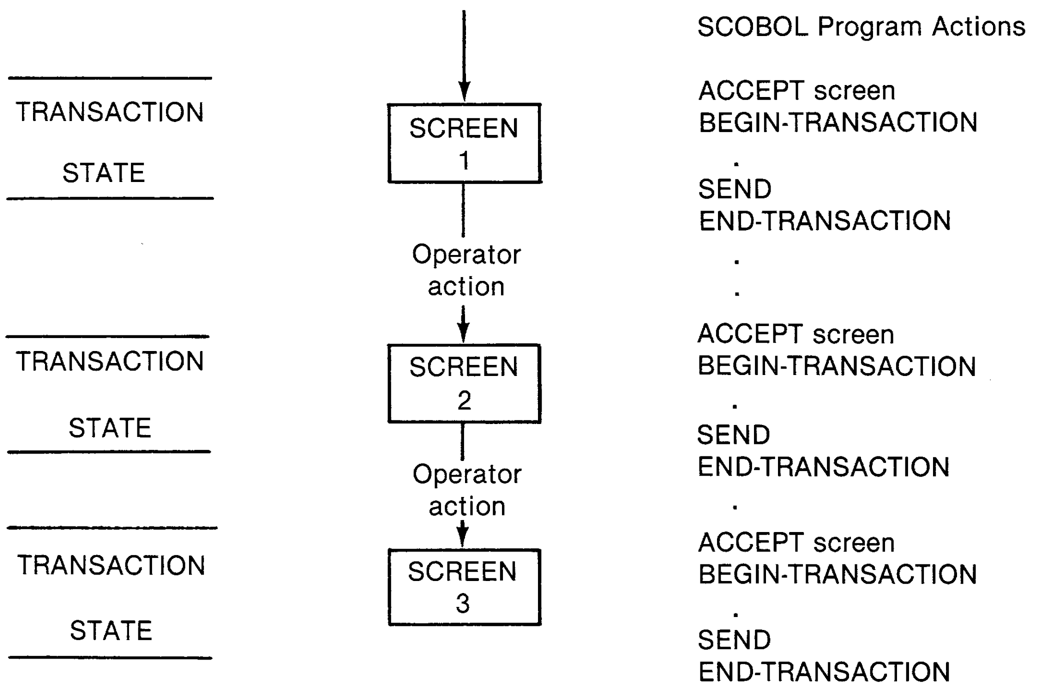


Advantages:
- Failure at any point in transaction state will cause all updates to audited files to be backed out.
- No operator action is required while resources are locked.

Disadvantages: Overuse of context could be a performance consideration.
- Increased TCP paging.
- Possibility of additional database access.
- Context stored in temporary disc files—implies more disc I/O, and more server activity.

Note: The new SCOBOL CHECKPOINT verb can be used to establish a restart point during the context gathering portion of the transaction.

MULTIPLE SHORT TRANSACTIONS. A typical sequence of Screen Cobol actions for multiple short transactions is illustrated below.



Advantages:
- Context use kept to a minimum.
- No operator action is required while resources are locked

Disadvantages:
- A logical transaction is possibly separated into several physical transactions.
- Backout of an entire logical transaction must be handled programmatically.
- Other transactions might view the database as inconsistent. This, also, must be handled programmatically.

NonStop Design Considerations for PATHWAY

In PATHWAY, properly written servers can be divided into three categories. These categories determine whether or not the server should be be NonStop.

1. Servers that access the database in read-only mode. Since all requests to servers of this type are retryable, they should not be NonStop.

2. Servers that update audited files. Because the transaction is a unit of recovery, this type of server should not be NonStop.

3. Servers that update unaudited files. There is no unit of recovery for servers of this type and they should be NonStop.

Summary of Design Considerations for PATHWAY

The following is a brief summary of the considerations that you follow to design PATHWAY applications that use TMF.

- Determine which database files are to be audited files.
- Determine the appropriate transaction design strategy for the application using the guidelines presented in the preceding part of this section. Decide where to put the BEGIN-TRANSACTION and END-TRANSACTION statements.
- Modularize the servers so that any server that updates audited files does not update unaudited files.
- Code the server to follow the TMF record locking rules.

INDEX

READER'S COMMENTS

Tandem welcomes your feedback on the quality and usefulness of its publications. Please indicate a specific *section* and *page* number when commenting on any manual. Does this manual have the desired completeness and flow of organization? Are the examples clear and useful? Is it easily understood? Does it have obvious errors? Are helpful additions needed?

Title of manual(s): ____________________

FOLD ➤

FOLD ➤

FROM:

Name ____________________

Company ____________________

Address ____________________

City/State ____________________ Zip __________

A written response is requested yes no

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 482 CUPERTINO, CA, U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

TANDEM
COMPUTERS

19333 Vallco Parkway
Cupertino, CA U.S.A. 95014
Attn: Technical Communications—Software

FOLD

STAPLE HERE

8063 A00
82063

TANDEM COMPUTERS INCORPORATED
19333 Vallco Parkway
Cupertino, CA 95014