

 **TANDEM** COMPUTERS

Transaction Monitoring in ENCOMPASS

Andrea Borr

Technical Report 81.2
June 1981

PN87601

Transaction Monitoring in ENCOMPASS

Andrea Borr

June 1981

Tandem Technical Report 81.2

Tandem TR 81.2

Transaction Monitoring in ENCOMPASS™.

Reliable Distributed Transaction Processing

Andrea Borr

Tandem Computers Incorporated
19333 Vallico Parkway, Cupertino Ca. 95014

June 1981

ABSTRACT: A transaction is an atomic update which takes a data base from a consistent state to another consistent state. The Transaction Monitoring Facility (TMF), is a component of the ENCOMPASS distributed data management system, which runs on the Tandem computer system. TMF provides continuous, fault-tolerant transaction processing in a decentralized, distributed environment. Recovery from failures is transparent to user programs and does not require system halt or restart. Recovery from a failure which directly affects active transactions, such as the failure of a participating processor or the loss of communications between participating network nodes, is accomplished by means of the backout and restart of affected transactions. The implementation utilizes distributed audit trails of data base activity and a decentralized transaction concurrency control mechanism.

Copyright © by IEEE. Originally appeared in Proceedings of Seventh International Conference on Very Large Databases, Sept 1981, IEEE Press. Republished by Tandem Computers Incorporated with the kind permission of IEEE.

CONTENTS

INTRODUCTION: Architecturally-derived data integrity vs. data base consistency	1
ARCHITECTURAL OVERVIEW	1
Hardware Architecture	1
The Tandem Operating System	2
DATA MANAGEMENT SYSTEM OVERVIEW	5
ENCOMPASS	5
Data Base Management	5
Terminal Management	6
Transaction Flow and Application Control	6
Transaction Management	7
TMF DESIGN OVERVIEW	10
Concurrency Control	10
Audit Trails	10
Transaction State Change	11
Distributed Transaction Processing	13
Distributed Commit Protocol	13
ROLLFORWARD	15
A DISTRIBUTED DATA BASE APPLICATION	16
CONCLUSIONS	18
ACKNOWLEDGEMENTS	18
REFERENCES	19

INTRODUCTION

The Tandem NonStop system architecture — hardware and software — is designed to provide failure-tolerance, expandability, and distributed data processing in an online transaction processing environment. The architectural overview which follows shows how such features as continuous availability, tolerance of single-module failures, fail-safe structural integrity of files, and I/O device fault tolerance derive from the design. The extension of the operating system to support a network of Tandem nodes is discussed. The network extension is reliable, highly available, and provides geographic independence. These features provide the foundation upon which to build a reliable distributed data management system; however, reliable distributed transaction processing requires that another layer of failure protection be provided for the data base. Logical data base consistency must be guaranteed despite processor failure, application process failure, network partition, transaction deadlock, or application-requested transaction abort. The means used by ENCOMPASS to provide these features are examined.

ARCHITECTURAL OVERVIEW

Hardware Architecture

The Tandem system is based on multiple, independent processors. Figure 1 illustrates the architecture of a typical three-processor system. The hardware structure consists of from 2 to 16 processor modules, each with its own power supply, up to two megabytes of memory, and I/O channel, interconnected by dual high-speed (13.5 megabytes/sec) interprocessor buses. Each I/O controller is redundantly powered and connected to two I/O channels. Disc drives may be connected to two I/O controllers, and discs themselves may be duplicated, or "mirrored", to provide data base access despite disc failures. At least two paths connect any two components in the system. Thus, hardware redundancy is arranged so that the failure of a single module does not disable any other module or disable any inter-module communication. Normally, all components are active in processing the workload. However, when a component fails, the remaining system components automatically take over the workload of the failed component ⁴.

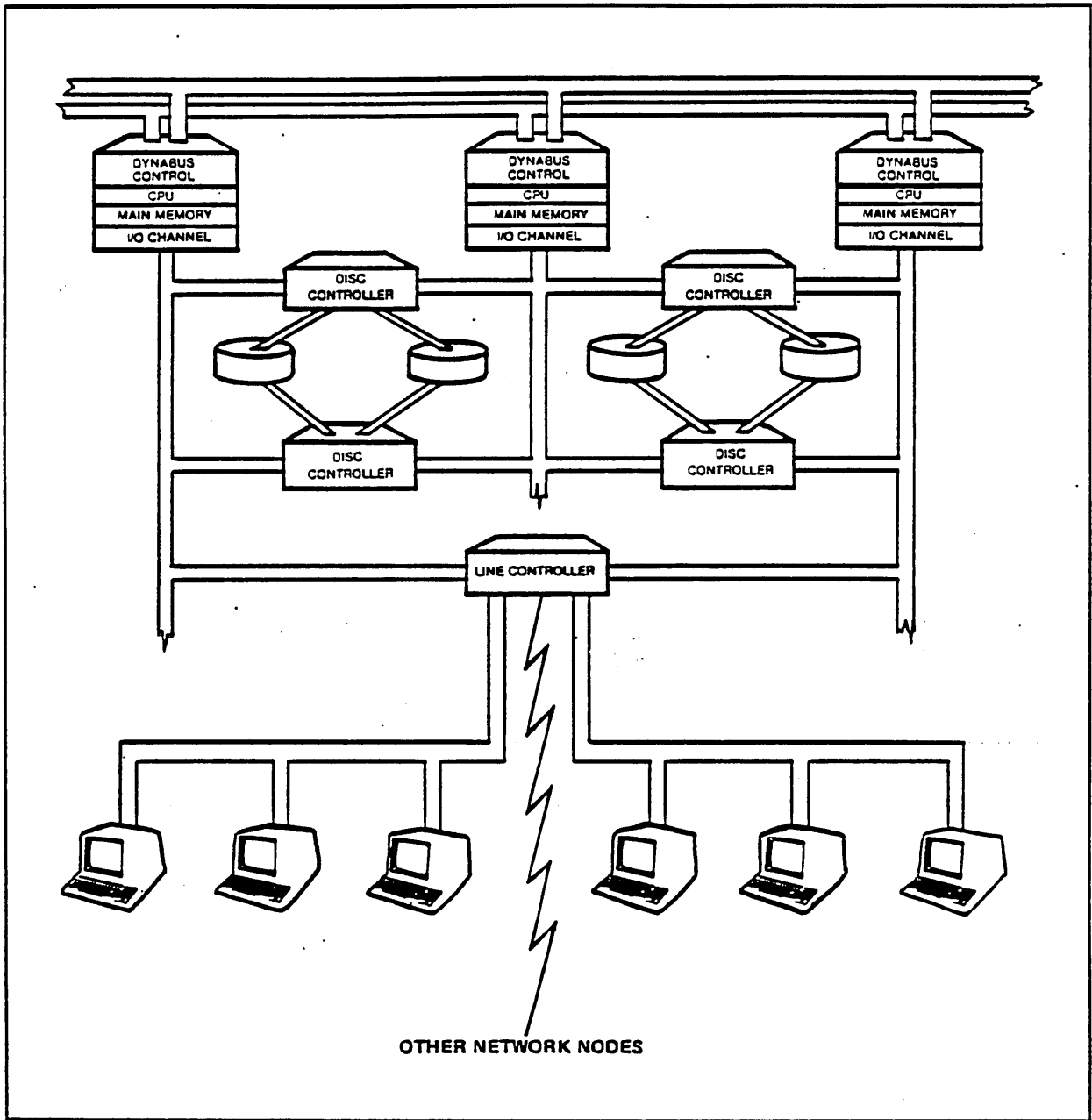


Figure 1. The Tandem Nonstop Hardware Architecture

The Tandem Operating System

System resources are managed by a message-based operating system which decentralizes information and control. The operating system resides in each component processor. The relationship among the processors of a system is characterized by symmetry and the absence of any master-slave hierarchy.

The operating system provides the software abstractions, processes and messages, necessary for decentralized control of the distributed components. All communications between processes is via messages. The Message System makes the physical distribution of hardware components transparent to processes. User processes access the Message System through the File System. The Message System and the File System effectively transform the multiple-computer structure into a unified multiprocessor at the user level.

System-wide access to I/O devices is provided by the mechanism of the I/O "process-pair". An I/O process-pair consists of two cooperating processes which run in the two processors physically connected to a particular I/O device. One of these processes, designated the "primary", controls the I/O device, handling all requests to perform I/O on the device. The other process, designed the "backup", functions as a stand-by in case of failure of the primary path to the device. The primary process sends the backup process "checkpoints", via the Message System, which ensure that the backup process has all the information that it would need in the event of failure to assume control of the device and carry through to completion any operation initiated by the primary. During normal operation, the backup is passive, acting only to receive the primary's checkpoints. In the event of an I/O channel error or a failure of the primary's processor, the backup process takes control of the device and becomes the primary.

The process-pair is a general mechanism utilized within the operating system to make system resources and services available to all processes in a fault-tolerant manner. An example of the use of the process-pair mechanism for a system process other than an I/O process is the "operator" process-pair, which is responsible for formatting and printing error messages on the system console. The primary and backup of the operator process run in two processors of the system. In the event of a failure of the primary's processor, the backup is able to continue offering this service. The continuous service provided by the process-pair is the essence of the feature termed NonStop.

The concept of the process-pair extends to application processes as well as to system processes. User-callable operating system routines are provided for creating a backup process and sending checkpoints to it. As shown by Bartlett ², a process-pair application can provide continuous fault-tolerant processing despite module failure.

The design of the Tandem operating system is described in more detail in [1] and [2].

The Tandem Network

The message-based structure of the Tandem operating system allows it to exert decentralized control over a local network of processors. Since it already addresses some of the problems of controlling a distributed computing system, the operating system has a natural extension to support a data communications network of Tandem nodes, each node containing up to 16 processors. (Henceforth, the terms "system" and "node" will be used interchangeably, and the term "network" will be used to refer to a collection of Tandem nodes connected by data communications links).

The extension of the operating system to the network operating system involves the generalization of message destinations to include processes in other nodes. This extension of the Message System beyond the boundaries of a single system allows a process anywhere in the network to send or receive a message from any other process in the network.

Features of the Tandem network include the following:

1. fault-tolerant nodes for high availability and data integrity;
2. user-level transparency of access to geographically distributed system resources;
3. decentralized control, characterized by the absence of a network master;
4. dynamic best-path message routing, including automatic re-routing in the event of a communications line failure;
5. automatic packet forwarding via an end-to-end protocol which assures that data transmissions are reliably received.

The design of the Tandem network system is described in more detail in [5].

DATA MANAGEMENT SYSTEM OVERVIEW

ENCOMPASS

The ENCOMPASS distributed data management system performs the functions required for the development and operational control of on-line application systems. The basic functions provided by ENCOMPASS components include: (1) data base management; (2) terminal management; (3) transaction flow and application control; and, (4) transaction management.

Data Base Management

The data base management component of ENCOMPASS provides a data definition language, a data dictionary, a relational data base manager, and a high-level non-procedural relational query/report language.

Among the features provided by the ENCOMPASS data base manager are the following:

1. three types of structured file organizations: key-sequenced, relative, and entry-sequenced;
2. multi-key access to records with automatic maintenance of the indices during file update;
3. data and index compression;
4. partitioning of files—by key value range—across multiple disc volumes (possibly on multiple nodes);
5. security controls by function, user class, network node, application program, and specified terminal;
6. a cache buffering scheme designed to keep the most recently referenced blocks of data in main memory.

The ENCOMPASS data base manager distributes data across multiple processors and discs, providing multiple points of control. Implemented as an I/O process-pair per disc volume, designated the DISCPROCESS, it protects the structural integrity of individual files through active checkpointing of process state and data, and recovery in the case of processor, I/O channel, or disc drive failure. The DISCPROCESS controls all access to a logical disc volume, which in the case of a mirrored device pair includes two physical disc drives and all primary and backup access paths (I/O channels and I/O controllers).

The DISCPROCESS protects the integrity of the files resident on its volume by maintaining control information and data in two processors—the processors in which the primary and backup DISCPROCESSes reside. If a failure occurs which prevents the primary DISCPROCESS from completing an operation it has started, the backup DISCPROCESS automatically takes over and completes the operation.

Two granularities of locking are provided for concurrency control: file and record. Record level locking operates on the primary key of an individual logical data record. (There is no locking at the block or index level.) Locks on existing records are obtained at read time by explicit application program request. All locks are exclusive mode.

Each DISCPROCESS maintains the locking control information for those records and files resident on its volume only. Thus, concurrency control for ENCOMPASS is decentralized and is effectively distributed among the DISCPROCESSes; no central lock manager exists. Deadlock detection is by timeout, the interval being specified as part of the lock request.

Terminal Management

The terminal management component of ENCOMPASS, known as the Terminal Control Process (TCP), provides screen formatting, data validation, screen sequencing, and data mapping. A TCP controls up to 32 terminals and supports a variety of terminal types and communication lines. The application interface to each terminal is defined by the user in a high-level language known as Screen COBOL (a COBOL-like language with extensions for screen handling). The user's Screen COBOL program is interpreted by the TCP to perform screen sequencing, data mapping, and field validation for a single terminal. A TCP supervises the interleaved execution of Screen COBOL programs, each associated with one of the terminals under control of the TCP. Multiple TCP's can be run to provide better distribution of available resources or to support large numbers of terminals.

TCP's are configured as process-pairs. As a result of the fault tolerance thus provided, the terminal user has continuous access to the executing Screen COBOL program despite module failure, including processor failure.

Transaction Flow and Application Control

ENCOMPASS applications have user-defined transactions that originate at terminals and access data bases. In addition to providing a Screen COBOL program defining the screen formats and controls for a terminal, the ENCOMPASS user provides a set of application program modules, known as application "server" programs, which access and update data base files. Screen COBOL programs and application server programs communicate by exchanging request and reply messages. Communication between a Screen COBOL program and a server is initiated by the Screen COBOL "SEND" verb. Under control of the SEND verb, the TCP, using the File System, passes a transaction request message to a server. The server performs the application function against the data base. The structure of an application server program is simple and single-threaded: (1) read the transaction request message; (2) perform the data base function requested; (3) reply. A server must be "context free" in the sense that it retains no memory from the servicing of one request to the next. Application servers are written to be independent of terminal and communications considerations. Servers may be written in available commercial languages such as COBOL.

ENCOMPASS application control provides monitoring of applications which are spread across a single system or network. It provides for the dynamic creation and deletion of application server processes to ensure good response time and utilization of resources as the workload on the system changes. Figure 2 shows a typical ENCOMPASS configuration.

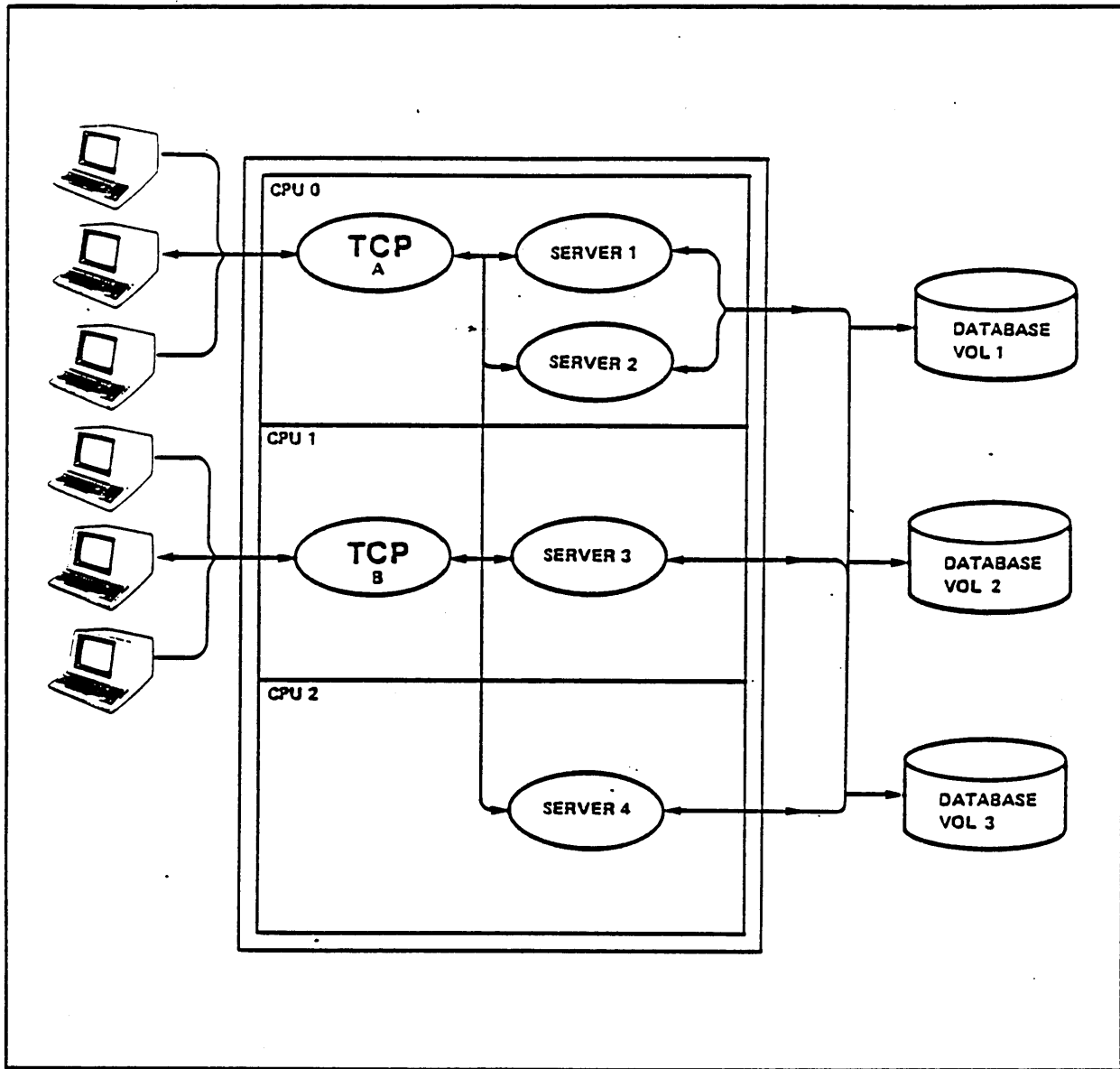


Figure 2. A Typical ENCOMPASS Configuration

Transaction Management

The transaction management component of ENCOMPASS is known as the Transaction Monitoring Facility (TMF). TMF implements the concept of a transaction as defined in the following.

A transaction is a (possibly multi-step) logical update which takes the data base from a consistent state to another consistent state. A consistent data base is one which satisfies an application-dependent set of assertions concerning the relationships between files, records, fields, and secondary indices, the truth of which is required for the data base to effectively model the application's world of reference.

In order for data base consistency to be maintained, a transaction must have the property of atomicity: either all of its effects persist or none of its effects persist. This in turn requires that a transaction which aborts for any reason, by user request or due to a failure, be backed out so that none of its effects persist. On the other hand, a transaction which completes reaches a point at which it abdicates the right to back out. At this point, the transaction is said to commit: all of its effects will persist regardless of subsequent failures.

Prior to the existence of TMF in ENCOMPASS, data base consistency had to be preserved by application fault-tolerance. The application had to be coded as a process-pair, which by careful design of checkpoint logic could recover correctly from single-module failures. (The run-time systems of COBOL and FORTRAN automated process-pair coordination and error-recovery to some extent.) Since process-pairs always carried forward to completion processing interrupted by failure, transactions always committed. The reversal of a transaction had to be coded by the user, as there was no automated transaction backout. Furthermore, there was no protection for the data base in the event of multiple-module failure. If a disc's primary and backup processors failed simultaneously, data on the disc could be left in an unrecoverable or inconsistent state.

The introduction of transaction backout by TMF makes it unnecessary to code the application as a process-pair. Without TMF, the application process must maintain correct state in a backup process in case the primary fails. With TMF, the state of progress of an incomplete transaction is immaterial, since failure will cause the transaction to be automatically backed out, restoring data base consistency. TMF further provides voluntary transaction backout, making it unnecessary for the user to code transaction reversal. Data base protection in the event of multiple-module failure is provided by the ROLLFORWARD facility described in a later section.

The ENCOMPASS user's interface to TMF is through use of the Screen COBOL verbs:

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

BEGIN-TRANSACTION is used to mark the beginning of a sequence of operations which should be treated as a single transaction. For the Screen COBOL program, BEGIN-TRANSACTION marks the beginning of a series of one or more SEND's of transaction request messages to application server processes. The network location of the application server process and, in fact, of the data base itself, is transparent to the Screen COBOL program. The server, the data base, or part of the database (in any combination) may reside on remote network nodes. For example, a server may transparently access data base files residing on any network node. A transaction may do work at multiple nodes and involve multiple server requests.

Execution of BEGIN-TRANSACTION causes a unique transaction identifier, or "transid", to be generated. The transid consists of a sequence number, qualified by the number of the processor in which BEGIN-TRANSACTION was called, qualified by the number of the network node which originated the transaction, designated the "home" node for the transaction. The Screen COBOL special register TRANSACTION-ID is set to contain the new transid, and the terminal is said to enter "transaction mode".

BEGIN-TRANSACTION marks a restart point in case of failure while the terminal is in transaction mode. If the transaction fails for any reason except an explicit ABORT-TRANSACTION by the Screen COBOL program (and the number of restarts has not exceeded a configurable "transaction restart limit"), the terminal's execution is restarted at BEGIN-TRANSACTION after TMF backs out any data base updates that have been performed for the current transid. A new transid is obtained for the new attempt at executing the logical transaction.

The types of failures which would result in the automatic abort, backout, and restart of a transaction by the system include: (1) failure of the primary TCP's processor (i.e. primary process of the process-pair); (2) failure of an application server's processor while that server was working on the transaction; (3) complete loss of communication with a network node which participated in the transaction. On the other hand, recovery from the failure of a component such as a primary DISCPROCESS' processor, an individual network communication line, a power supply, I/O controller, or disc drive is handled automatically by the operating system transparently to transaction processing.

The effect of a processor or other single module failure, which would necessitate crash restart and data base recovery on a conventional system, is limited to the on-line backout of those transactions in process on the failed module. Transactions uninvolved in the failure continue processing. Because the TCP checkpoints data extracted by the Screen COBOL program from input screen(s) to its backup, in many cases the restart of a logical transaction may not require re-entering the input screen(s).

All SEND's executed by a Screen COBOL program whose terminal is in transaction mode have the terminal's current transid automatically appended to the interprocess message by the File System. When the application server reads the transaction request message, the terminal's current transid becomes the "current process transid" for the application process. When the application process then executes a statement requiring disc I/O and/or record or file locking, the File System automatically appends the application process' current transid to the request message which is sent to the DISCPROCESS.

When all the SEND's required for the transaction's execution are complete, the Screen COBOL program indicates that the transaction should be committed by executing the END-TRANSACTION verb. At the completion of the execution of this verb, the transaction's data base updates become permanent and will not under any circumstances be backed out. The Screen COBOL program's END-TRANSACTION request can, however, be rejected because the transaction has been aborted by the system due to one of the causes of automatic abort, e.g. network partition. If so, the Screen COBOL program may be restarted at the BEGIN-TRANSACTION point.

If the Screen COBOL program, or any of the servers to which it has done a SEND, detects a need to abort and back out a transaction—without automatic restart by the TCP—the Screen COBOL program executes the ABORT-TRANSACTION verb.

Finally, RESTART-TRANSACTION is used to indicate that the current attempt to execute the transaction has failed due to a transient problem and so should be backed out and restarted. For example, a server may request a data base record lock with a timeout interval specified; then, in case the timeout occurs, it would recover from a possible deadlock by replying to the SEND with an error result indicating that the Screen COBOL program should call RESTART-TRANSACTION.

TMF DESIGN OVERVIEW

Concurrency Control

Gray defines a transaction that sees a consistent data base state as one that (a) does not overwrite dirty data of other transactions; (b) does not commit any writes until the end of transaction; (c) does not read dirty data from other transactions; and (d) prior to its completion, does not permit any data it reads to be dirtied by other transactions. If all transactions observe these protocols, then transaction backout produces a consistent state ³.

TMF enforces clauses (a), (b), and (c) as follows. It verifies that all records updated or deleted by a transaction have been previously locked by that transaction. (A lock on an existing record is acquired at record read time by explicit application request.) TMF automatically generates locks on all new records inserted by a transaction and on the primary key values of all records deleted by a transaction. Clause (d) insures that the reads performed by a transaction are repeatable ³. The observance of clause (d) is recommended to writers of TMF transactions, but for system performance reasons is not enforced, as enforcement would require the generation of a lock for each record read by a transaction.

Audit Trails

TMF maintains distributed audit trails of logical data base record updates on mirrored disc volumes. An audit trail is a numbered sequence of disc files whose volume of residence is configurable and whose creation and purging is managed by TMF. The locations of audit trails for disc volumes containing data base files designated by the user as "audited" are independently configurable. Each DISCPROCESS which manages a disc volume configured as "audited" (i.e. capable of containing audited data base files) automatically provides "before-images" and "after-images" of data base updates by application processes to an AUDITPROCESS (of which several, each a process-pair, are configurable), which writes to an audit trail. All audited discs on a given controller share an AUDITPROCESS and an audit trail. Multiple controllers may be configured to use the same or different AUDITPROCESSES and audit trails. Auditing of data base updates is totally transparent to application programs. For transactions that span data bases on multiple nodes of a network, all audit images for records residing on a particular node are contained in audit trails at that node. This enables transaction backout at a node to occur without the need for communication with other nodes. Transaction backout is performed by the BACKOUTPROCESS (a process-pair), using the transaction's before-images recorded in the audit trails.

The implementation of the DISCPROCESS as a process-pair, residing in two processors, eliminates the necessity for the protocol, termed "Write Ahead Log" by Gray ³, which requires before-images to be write-forced to the audit trail prior to performing any update of the data base on disc. The Write Ahead Log protocol enables restart after system crash using conventional recovery techniques. In the NonStop approach to handling failure, checkpoint is the functional equivalent of Write Ahead Log. By checkpointing the audit records generated by an update request to its backup prior to performing the update, the primary DISCPROCESS assures the feasibility of transaction backout even in the event of the failure of the primary's processor. As in Write Ahead Log, however, all audit records generated by a transaction are write-forced to disc as part of the two-phase transaction commit protocol.

In addition to the data base audit trails, TMF maintains, at each node, a "Monitor Audit Trail" which contains a history of transaction completion statuses: commits and aborts. A transaction commits at the time its commit record is written to the Monitor Audit Trail.

Transaction State Change

A transaction goes through several state changes during the commit or abort protocol. All transaction state changes are broadcast, via the interprocessor bus, to all processors within a single node. This is done regardless of which processors actually participated in the transaction. In contrast, in the network case only those nodes which participated in the transaction are notified of transaction state change. The decision to broadcast transaction state change information to all processors within a single system was taken because of the speed and reliability of the interprocessor bus as a communication medium.

A summary of the possible states of a transaction follows:

1. **Active.** A transaction has this state after BEGIN-TRANSACTION has been called but before commit or abort has been requested. BEGIN-TRANSACTION broadcasts the transid in "active" state to all processors in the system. The possible states which can follow are "ending" or "aborting".
2. **Ending.** A transaction has this state after END-TRANSACTION has been called but before the transaction commit record has been written to the Monitor Audit Trail. During "ending" state, all the transaction's audit records are written to the audit trails. This constitutes "phase one" of transaction commit. The possible states which can follow are "ended" or "aborting".
3. **Ended.** A transaction has this state after the transaction commit record has been written to the Monitor Audit Trail. Once the transaction has entered the "ended" state, the Screen COBOL END-TRANSACTION verb completes, and participating DISCPROCESSes are notified to release the committed transaction's locks. This constitutes "phase two" of transaction commit. Once the "ended" state has completed, the transid leaves the system.
4. **Aborting.** A transaction has this state after the decision to back out the transaction has been taken, but before any of its locks are released. While the transaction is in "aborting" state, all of its audit records are written to the audit trails and the transaction's data base updates are backed out by the BACKOUTPROCESS. "Aborting" and "ending" are parallel states. The only possible following state is "aborted".
5. **Aborted.** A transaction has this state after the transaction has been backed out. Once the transaction has entered "aborted" state, participating DISCPROCESSes are notified to release the backed out transaction's locks. "Aborted" and "ended" are parallel states. Once the "aborted" state has completed, the transid leaves the system.

The state transitions of a transaction are illustrated in Figure 3.

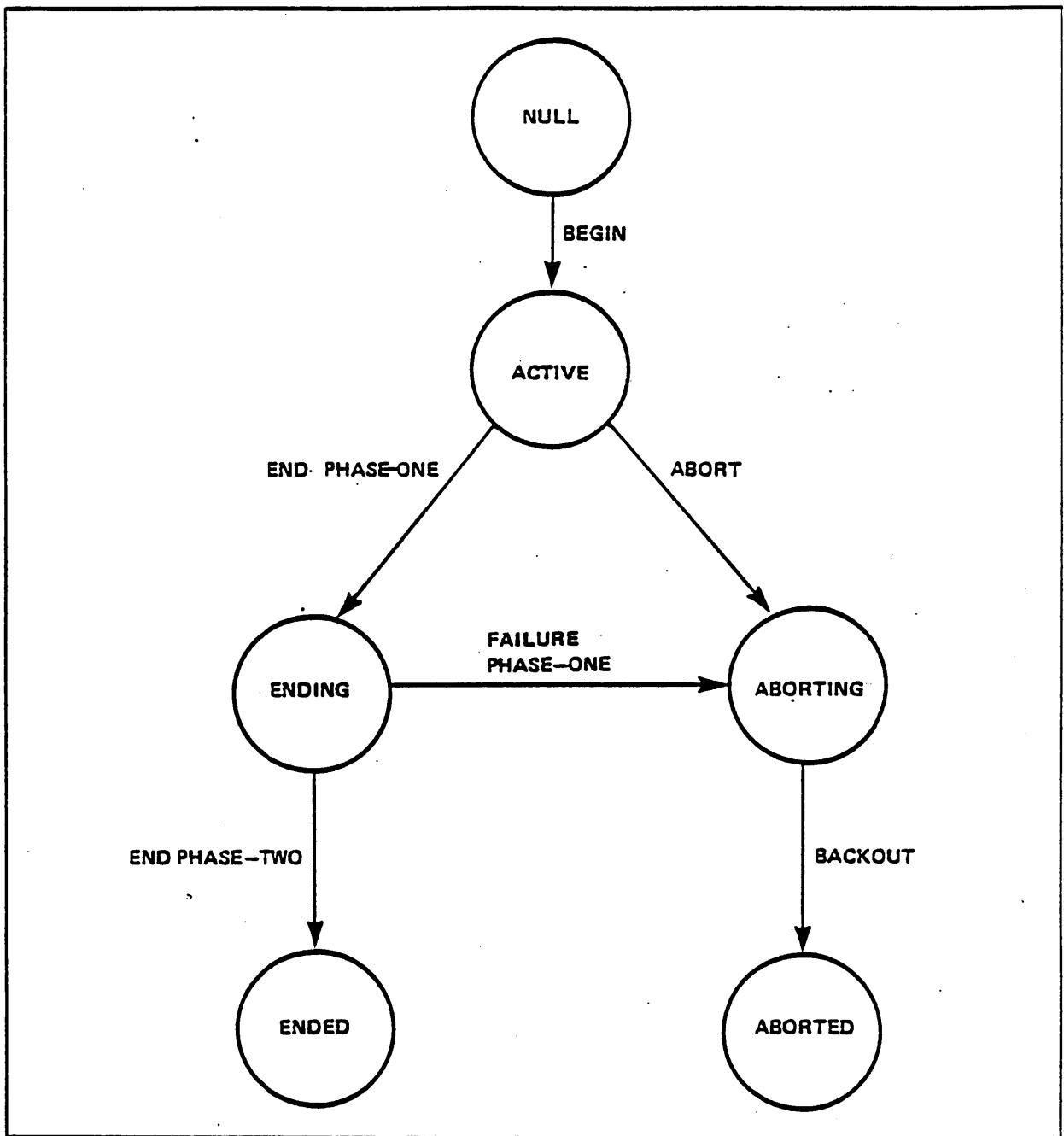


Figure 3. State Transitions of a Transaction

For transactions which stay within a node, TMF uses an abbreviated two-phase commit protocol. Its purpose is to ensure that all audit records generated by a transaction are written to disc prior to unlocking the transaction's locks, thus making the transaction's output visible to concurrent transactions. This guarantees that a transaction, once committed, can always be recovered using ROLLFORWARD, a TMF utility which can be used to apply after-images from the audit trails to a previously archived copy of the data base. ROLLFORWARD is discussed in a later section.

Distributed Transaction Processing

Transactions can originate on any node of the network and can be transmitted by any path to any number of other network nodes through SEND's from a TCP to remote servers or I/O requests from servers to remote discs. TMF ensures data base consistency both within a single node and across nodes by treating all data base updates performed by a transaction as a group, identified network-wide by the transid. All of the updates in the group are made permanent by the execution of the Screen COBOL END-TRANSACTION verb. If a failure occurs, the transaction's updates are backed out on all participating nodes.

The strategy, used in the single-node case (up to 16 processors), of broadcasting transaction state change information to all processors, regardless of their participation in the transaction, is clearly too expensive to use in the network case. Furthermore, the information is likely to be useless to most of the network. Therefore, only nodes participating in the transaction receive state change information.

Coordination of distributed transactions is one of the functions of the "Transaction Monitor Process" (TMP), a process-pair which is configured for each network node that participates in the distributed data base. Whenever a transid is transmitted by the File System to a remote node (as a result of a SEND to a remote server or an I/O request to a remote audited data base file), the TMP on the sending node determines whether the destination node has received a previous transmission of the requesting transid from the sending node. If not, the TMP on the sending node notifies the TMP on the destination node to broadcast the transid in "active" state to all processors on its node. This "remote transaction begin" occurs prior to any transmission of the transid by the File System to a server or DISCPROCESS on the destination node.

Distributed Commit Protocol

In a distributed data base environment, where a single transaction may result in the update of files on multiple nodes, loss of communication between nodes may cause data base inconsistency. TMF uses a more elaborate two-phase commit protocol for distributed transactions than that used for single-node transactions, due to the unreliability of the communication medium and the loose coupling of the nodes. The distributed commit protocol allows any participating node to unilaterally abort a transaction. The purpose of phase one is twofold. First, it serves to ensure that all audit records generated by a transaction are written to the audit trails on all participating nodes prior to allowing the unlocking of any of the transaction's locks. Secondly, it guarantees that the decision to commit or abort a transaction is uniform across all nodes, even in the event of loss of communications between participating nodes or the catastrophic failure of a node.

In the distributed case, transaction state change is accomplished by TMP-to-TMP messages sent over the network. Each participating node sends transaction state change messages to the TMP on all nodes for which it was the direct source of transid transmission. On receipt of a transaction state change message, the TMP broadcasts the state change to all processors within its node.

Certain network TMP message types are termed "critical response". For critical response messages, the destination TMP must be accessible at the time the message is initiated, and it must reply with an affirmative response in order for the transaction state change to proceed. Examples of critical response messages to remote TMP's are the network message requesting remote transaction begin and that requesting phase one of commit (i.e. transaction state change to "ending").

Other network TMP message types are termed "safe-delivery". For safe-delivery messages, the destination TMP need not be accessible at the time the message is initiated, and the reply serves only to acknowledge receipt of the message rather than to signify concurrence. The sending of safe-delivery messages—whenever transmission becomes possible—is guaranteed, but their delivery is not time-critical to the transaction's state change. Examples of safe-delivery messages to remote TMP's are the network message requesting transaction state change to "ended" (phase two of commit) and that requesting transaction state change to "aborting" (requesting transaction backout). Following transaction backout, state change to "aborted" on each participating node is accomplished under local control, without need for further network communication.

The critical response message requesting phase one of commit is initially transmitted over the network by the TMP on the transaction's home node in response to the Screen COBOL program's END-TRANSACTION call. For phase one to complete successfully, each node to which the home node directly transmitted the transid must be accessible and must reply affirmatively after having written the transaction's audit records to disc on its node and after having assured, transitively, that all nodes to which the transid was further transmitted have done likewise. The existence of a node participating in the transaction which is either inaccessible at phase-one time, or which responds negatively to the phase-one request because it has previously decided to abort the transaction (e.g. due to a prior loss of communications) will cause the commit attempt to fail. The TMP on the transaction's home node will transmit transaction backout messages in this case. The successful completion of phase one, on the other hand, will cause the home node TMP to transmit phase-two messages, causing the release of the transaction's locks throughout the network and the completion of the END-TRANSACTION call.

For example, suppose a TCP on node 1 SENDs to a server on node 2, which in turn updates a record via a DISCPROCESS on node 3. The TMP on node 1 remembers that it transmitted the transaction to node 2, but does not know that node 2 transmitted it to node 3. The TMP on node 2 remembers that it transmitted the transaction to node 3. When END-TRANSACTION is called on node 1, "ending" state is broadcast to all processors on node 1. The TMP on node 1 receives this broadcast and sends a network message to the TMP on node 2. The latter broadcasts to all processors on node 2 and in addition sends a network message to the TMP on node 3, which broadcasts to all processors on its node. This causes the transaction to go into "ending" state on all processors of all participating nodes. This is the first phase of the commit protocol. The second phase is similar.

Once phase one has completed successfully, the inability to communicate with all participating nodes during phase two (lock release) does not impede the completion of the Screen COBOL program's END-TRANSACTION call on the home node. It merely means that records locked by the committed transaction on inaccessible remote nodes will remain locked until such time as communication is restored.

Until a non-home node has replied affirmatively to the phase-one message, it can unilaterally abort the transaction, and then force network consensus to abort by replying negatively to the phase-one message when it is received. Once a non-home node has replied affirmatively to the phase-one message, however, it must hold the transaction's locks until notification of the transaction's final disposition (i.e. to state "ended" or to state "aborting") is received (possibly indirectly) from the transaction's home node. If communication is lost at this point, the transaction's locks on the inaccessible node will be held until communication is restored. The manual override for this situation requires the following steps: (1) use of a TMF utility on the home node to determine the transaction's disposition; (2) a telephone conversation (for example) between operators on the home node and on the inaccessible non-home node; and, finally, (3) use of the TMF utility on the non-home node to force like disposition of the transaction.

ROLLFORWARD

Conventional transaction management systems must be optimized for quick restart in the event of total node failure. NonStop systems allow optimization of normal processing at the expense of restart time. For example, audit records need not be written to disc prior to updating the data base. On the other hand, however rare the occurrence of total node failure (e.g. simultaneous failures of two processors hosting a process-pair), TMF must have a provision for recovering the data base.

TMF's approach to recovery from total node failure is based on occasional archived copies of audited data base files, plus an archive of all audit trails written since the data base files were archived. These copies can be created during normal transaction processing. TMF reconstructs any files open at the time of a total node failure by using the after-images from the audit trail to reapply the updates of committed transactions. ROLLFORWARD negotiates with other nodes of the network about transactions which were in "ending" state at the time of the node failure.

A DISTRIBUTED DATA BASE APPLICATION

Tandem's Manufacturing Division uses ENCOMPASS to implement a reliable distributed data base to coordinate its four manufacturing facilities in Cupertino (California), Santa Clara (California), Reston (Virginia), and Neufahrn (West Germany). Manufacturing is one application of Tandem's 50-node corporate network. (The network grows at a rate of about two nodes per month; the manufacturing application adds approximately two nodes yearly.) The network for the manufacturing system is shown in Figure 4.

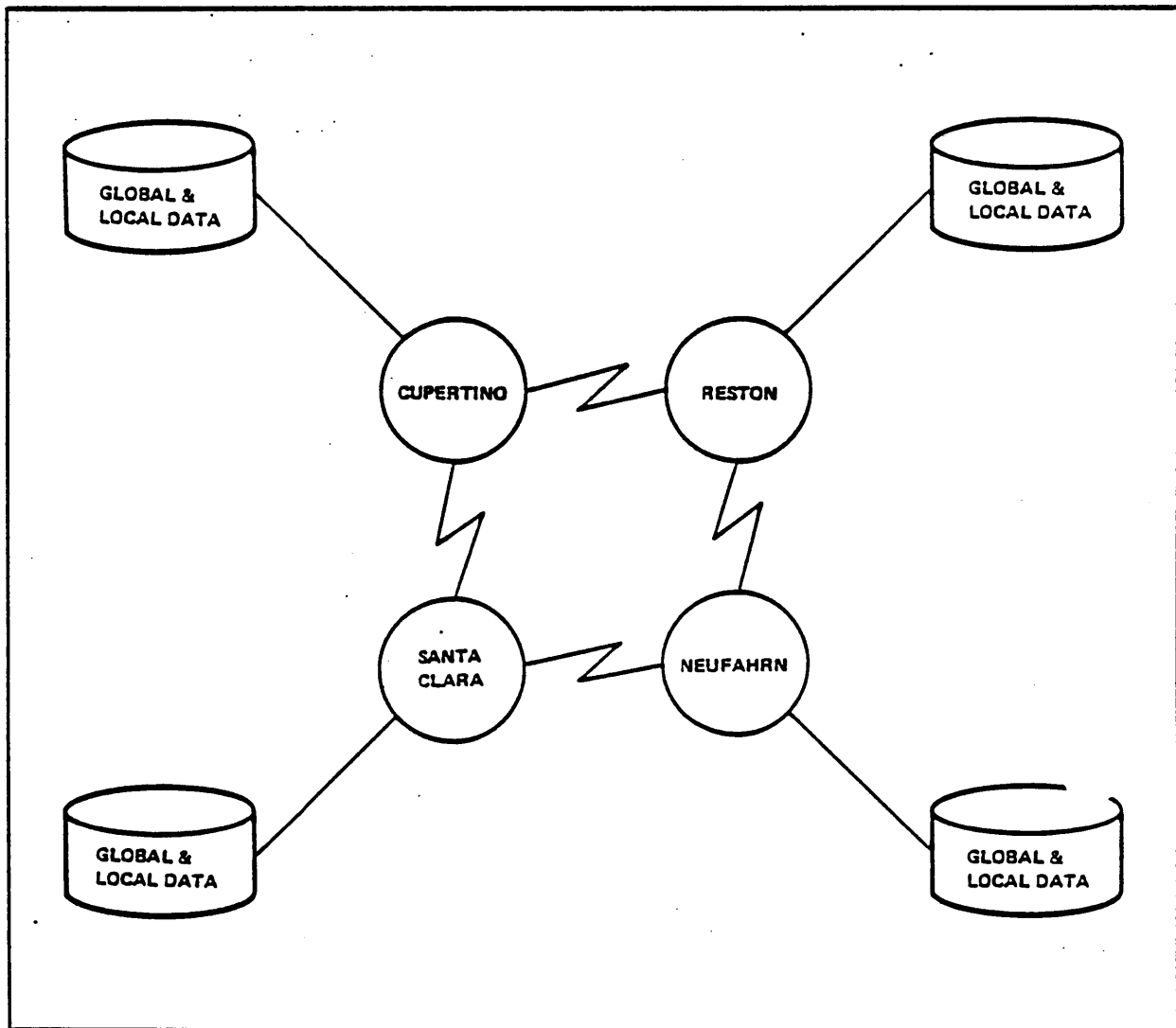


Figure 4. Manufacturing Network

Each node has a copy of the "global" files: Item Master File, Bill of Materials File, and the Purchase Order Header File. In addition, each node has a set of "local" files: Stock File, Work-in-Progress File, Transaction History File, and the Purchase Order Detail File.

The global files are replicated at all nodes for reasons of performance and availability. Most transactions access and update only local files. Transactions which reference global files access only a few records and occur infrequently. Transactions which update global files can originate at any node.

The designers of the system were faced with two conflicting goals: (1) the maintenance of consistency among the global file copies; and (2) node autonomy: the ability for a node to carry on its processing, including the update of global data, despite network partition or the unavailability of other nodes.

If consistency were the only goal, straightforward application of TMF to the problem would have been the solution. All global files would be TMF-audited. All reads of a record in a global file would be directed to the local copy. Updates of global files would be applied to all copies, within the scope of a single TMF transaction. Unfortunately, this simple approach fails to address the goal of node autonomy, since no node can run a global update transaction at a time when any other node is unavailable.

The actual design compromises the goal of replica consistency somewhat for the sake of node autonomy. As in the above design, all files are TMF-audited and reads are always directed to the local record copy. For the purpose of update, however, each global file record is assigned a master node, the name of which is stored in each record instance. The update of a global record can occur only if its master node is available. An update request is sent to a server on the record's master node. The server executes a TMF transaction which updates the master copy of the record and queues "deferred" update requests for the non-master copies of the record in a "suspense file" at the record's master node. A dedicated process, called the "suspense monitor", scans the suspense file looking for work to do. When it finds a deferred update record for a node which is currently accessible, the suspense monitor executes a TMF transaction which sends the update to a server at the non-master node and deletes the suspense file entry.

It is important that the deferred updates for non-master copies of records at a node occur in suspense file order. If a node becomes disconnected from the network, deferred updates for it accumulate in the suspense files of other nodes, and the disconnected node's suspense file accumulates updates for other nodes. When the network is re-connected and all accumulated updates are applied, global file copies converge to a consistent state.

CONCLUSIONS

The design of TMF is seen to rely heavily on the concept of NonStop, Tandem's unique approach to fault tolerance which provides continuous operation despite single module failures. Unlike conventional data base recovery techniques, which are oriented to repairing the data base after a system halt and restart, TMF maintains data base consistency through failures via the on-line backout of those individual transactions affected by the failure.

Since a Tandem node is in itself a local network of up to 16 processors, the techniques used to implement such features as transaction concurrency control and transaction commit coordination within a single node are of necessity oriented to the distributed environment. Consequently, their extension to a network is a relatively straightforward extrapolation of the single-node implementation. The differences between the handling of the single-node case and the network case are largely accounted for by the significant disparity in the speed and reliability of the communication media.

ACKNOWLEDGEMENTS


I would like to acknowledge the contributions of Glenn Peterson, Keith Hospers, Bill Earl, Gary Kelley, Keith Stobie, Jerry Held, Dennis McEvoy, and Glenn Linderman in the design, implementation, and testing of TMF. The design of Tandem's manufacturing data base is due to Jim Gray and Vince Sian. Thanks are furthermore due to Jim Gray for editorial suggestions whose implementation improved the presentation of this material.

REFERENCES

1. Bartlett, J. F., *A 'NonStop' Operating System*, Eleventh Hawaii International Conference on System Sciences, 1978.
2. Bartlett, J. F., *A NonStop Kernel*, Proceedings of Eighth Symposium on Operating Systems Principles, ACM, 1981 (also Tandem TR 81.4).
3. Gray, J. N., *Notes on Data Base Operating Systems*, IBM Research Report: RJ 2188, February 1978.
4. Katzman, J. A., *A Fault-Tolerant Computing System*, Eleventh Hawaii International Conference on System Sciences, 1978.
5. Katzman, J. A., and Taylor, R. H., *GUARDIAN/EXPAND, a NonStop Network* (submitted for publication).
6. Smith, L., *Designer's Overview of Transaction Processing*, (submitted for publication).

Copyright © 1981 by Tandem Computers Incorporated

ENCOMPASS, Tandem, and NonStop are trademarks of Tandem Computers Incorporated.

Distributed by
 **TANDEM COMPUTERS**
Corporate Information Center
19333 Vallco Parkway MS3-07
Cupertino, CA 95014-2599

