TANDEMCOMPUTERS

EMPACT: A Distributed Data Base Application

Alan Norman
Mark Anderton

Technical Report 83.1
October 1,1982
PN87605

EMPACT: A DISTRIBUTED DATA BASE APPLICATION

Alan Norman and Mark Anderton

October 1, 1982

Tandem Technical Report 83.1

EMPACT™: A Distributed Data Base Application

Alan Norman and Mark Anderton
Tandem Computers Incorporated
19333 Vallco Parkway, Cupertino, CA 95014

October 1, 1982

ABSTRACT: EMPACT, Tandem's Manufacturing Information Control System, is an application that uses a distributed database. The requirements of the system include supporting multiple sites, providing continuous availability to the data, and controlling updates to communal information. The approach taken to satisfy these needs involves the use of both partitioned and replicated data. Presented in this paper is a discussion of the application requirements, the design and architecture of the database, and the algorithms used for updates and inserts.

CONTENTS

INTRODUCTION

EMPACT, Tandem's manufacturing information control system, is an application that makes use of a distributed database. The requirements of the system include:

1. Supporting multiple sites of independently managed machines.
2. Providing the continuous availability of communal information.
3. Permitting both decentralized control of the application and management of the data.
4. Allowing for centralized development of the application.

This paper discusses the application requirements, the design and architecture of the database, and the methods used for maintaining the database.

APPLICATION REQUIREMENTS

Application Functions

EMPACT was developed to satisfy the data processing requirements of Tandem's manufacturing division. It was initially designed as a centralized online application that would support a single manufacturing site. The functions contained within EMPACT pertain to all the standard assembly manufacturing needs:

- Parts Master
- Bills of Material
- Inventory Control
- Purchasing/Receiving
- Work in Process
- Master Scheduling
- Materials Requirement Planning
- Inter-plant Material Transfer (MART)
- Job/Lot Tracking

Database Size and Service Levels

EMPACT was implemented on a Tandem T/16 computer using Tandem's distributed data management system, ENCOMPASS. The database size and the required service levels for the initial design were moderate:

1. Single manufacturing site.
2. Database of approximately 120 megabytes.
3. Database containing 500 + data elements and over 40 record types.
4. A user community of 150 persons using 55 terminals.
5. Monthly transaction rates: 15,000 against the entire database, with 4,500 update transactions against the bill of material files and 1,000 against the item master file.
6. A 6:1 ratio of reads to writes against the item master and bill of materials files.
7. Hardcopy batch reports generated nightly to supplement the online inquiry capabilities; these reports were created using ENFORM, Tandem's query language.

Business Environment

Tandem, as a company, anticipated high growth rates over the next several years. This meant that the size of the user community, the transaction volume, and database requirements were expected to increase accordingly. Furthermore, the manufacturing management decided to decentralize its operations. The plan was and is to disperse the manufacturing effort into several plants, all performing similar operational functions although geographically distant. A growth rate of one new plant every nine months was predicted.

This need to support geographically distributed sites placed some new demands upon EMPACT. As a business requirement, communal data such as item master or bill of materials information would have to be continuously available; the day-to-day business of the individual sites is highly dependent upon this information. Also, because of the decentralized management structure of Tandem's manufacturing organization, the sites required as much autonomy as possible; the daily business of one site could not adversely affect or be affected by events or circumstances at other sites. Each site was also to be responsible for and have control over the data stored there.

Initial Development Approach

The conceptual approach taken was to divide the elements of the database into two major categories, global data and local data, and one additional category that must be acknowledged but does not enter into this analysis, private data.

Global data are information that would be common to and shared by all sites. Examples of global data are the list of parts that determine Tandem's parts catalog (item master file) or the records that describe Tandem's product structure (bill of material file).

Local data are information that is uniquely important to the individual site using it but accessible by all sites. Examples of local data are stock status and work-in-process data. Local data has the same format as the corresponding data at other sites.

Private data are information that is used only by the individual site and is not visible to other sites. This data consists of reports and data outside the EMPACT system.

Global and local data are segregated into record units that are themselves either global or local. This segregation is not a requirement of the distributed application but a means of simplifying certain aspects of the application.

Figure 1 shows the division of the EMPACT database into global and local portions. As previously stated, the major requirement addressed by distributed EMPACT is site responsibility for the maintenance of local information and providing equal shares of control and responsibility for the maintenance of global information.

Global data are necessarily replicated because of the requirement for site independence. If global data were not replicated at all EMPACT sites, then disconnecting any nonglobal site from the network would mean that business activities at that site would be interrupted until the site rejoined the network. Any scheme of centralized global data suffers from this unacceptable possibility.

By our early reckoning, data would be either global (replicated at all sites) or local (single-site resident). However, another condition surfaced that called for only partial replication of data. The interplant material transfer function, known by the acronym MART, permits a request by one site for material from another to be placed and processed, and monitors the subsequent transfer of material. The process requires that all data and status information pertaining to the request be resident at both sites. However, the information is not of interest to any third party. Such data is classified as semiglobal since it requires only partial replication.

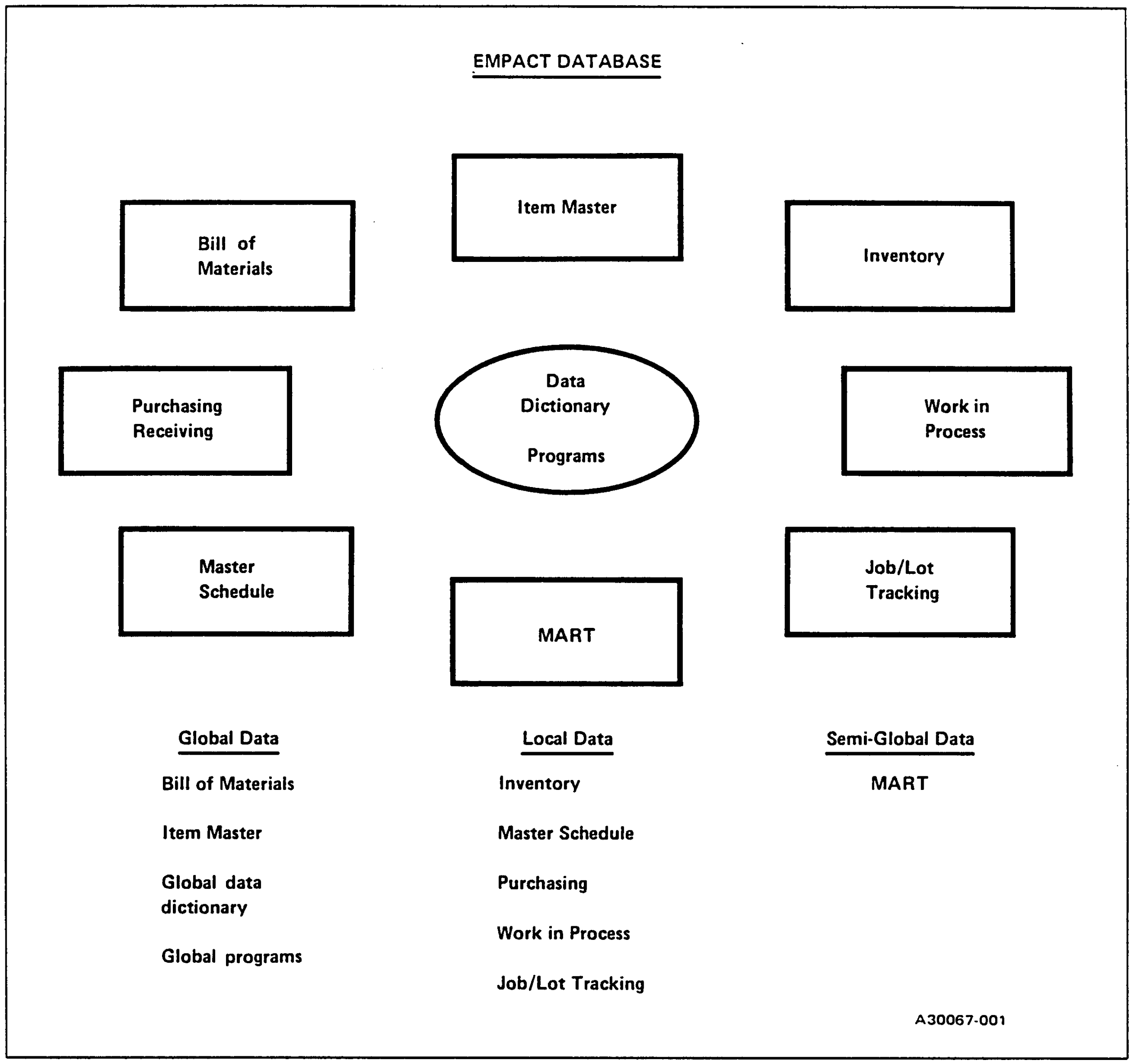


Figure 1. Structure of EMPACT database.

The next section provides a brief description of the Tandem software products that are the foundation of distributed EMPACT.

TANDEM'S SOFTWARE ENVIRONMENT—AN OVERVIEW OF ENCOMPASS-EXPAND

The expansion of EMPACT to support multiple sites was implemented using ENCOMPASS, Tandem's data management system, and EXPAND, Tandem's network software. ENCOMPASS is a conglomerate of several major software products – ENSCRIBE, DDL, ENFORM, PATHWAY, and TMF, the transaction and recovery manager.

ENSCRIBE is a relational database manager that supports three types of structured file organizations: key-sequenced, relative, and entry-sequenced. Within these file types, data are organized into logical records. EMPACT restricts itself to fixed-length records and key-sequenced files.

DDL is a data definition language used to define the data elements and record structures that make up an application. The output generated by DDL is stored in a data dictionary that is used by ENFORM, a high-level nonprocedural query language, to generate reports against the application database.

PATHWAY provides a terminal-oriented interface for transaction design and control. Its primary components are a terminal control process (TCP) that provides screen formatting and control, user written programs, known as *servers* which access and update the data files using ENSCRIBE procedures, and PATHMON, a monitor process that dynamically manages and allocates application resources.

An application's interface to the end-user is a set of program modules, or *requesters* , that are written in Screen-COBOL (a COBOL-like language with extensions for screen manipulation). The Screen-COBOL programs are interpreted by the TCP to perform screen sequencing, data mapping, and transaction control. The Screen-COBOL programs communicate with the servers by exchanging request and reply messages, hence the names requester and server. The servers are single-threaded programs that:

1. Read a request message.
2. Perform the database function requested.
3. Reply.

They must be *context-free* meaning that they do not need a memory of past requests in order to execute the current request.

A server is known to the requester programs by its *server class name*. A *server class* is a group of server programs that perform identical functions. When a TCP needs to establish communication with a particular server class, it requests a *link* to an individual server within the given server class from PATHMON (see Figure 2). PATHMON responds by sending the TCP the process name of an already existing server or of a newly created one. How PATHMON decides to respond depends on parameters established when the system is started up. This is one aspect of the load balancing and resource management that PATHMON performs.

Transactions in ENCOMPASS are defined as multistep operations that change the database from one consistent state to another. They are processed via the requester/server relationship. The end-user initiates a transaction request at a terminal; the requester module responds by sending a series of one of more requests to a set of servers; the servers perform the application function against the database; replies are returned to the requester; and the end-user is notified of the result at the terminal. Smith discusses the requester/server relationship in more detail.[4]

TMF guarantees the consistency of the database by ensuring that precisely all or none of the changes that a transaction attempts to make are permanent. For a more in depth discussion on TMF, refer to Borr's report.[1]

EXPAND, Tandem's network software, has several features relevant to application design. Control is decentralized; it is characterized by the lack of a network master. Access to geographically distributed system resources is transparent to the system user. In fact, local and remote requests to servers or files look the same to those programs initiating the requests. Nodes in an EXPAND network are defined by their system name, for example, \LA or \SANFRAN. Interprocess communication across the network requires only that the system name be specified, all other aspects are managed by the operating system. The design of the EXPAND network is described in more detail by Katzman and Taylor.[2]

ENVIRONMENT—EMPACT

EMPACT PATHWAY Configuration

EMPACT, prior to becoming distributed, had over 120 requesters and 40 server classes. There were over 200 transaction types and approximately 100 different reports. Typically, each subsystem, for example, bill of materials would have three server classes associated with it: one for updates, one for online queries, and one for report generation. Figure 2 shows the EMPACT PATHWAY configuration.

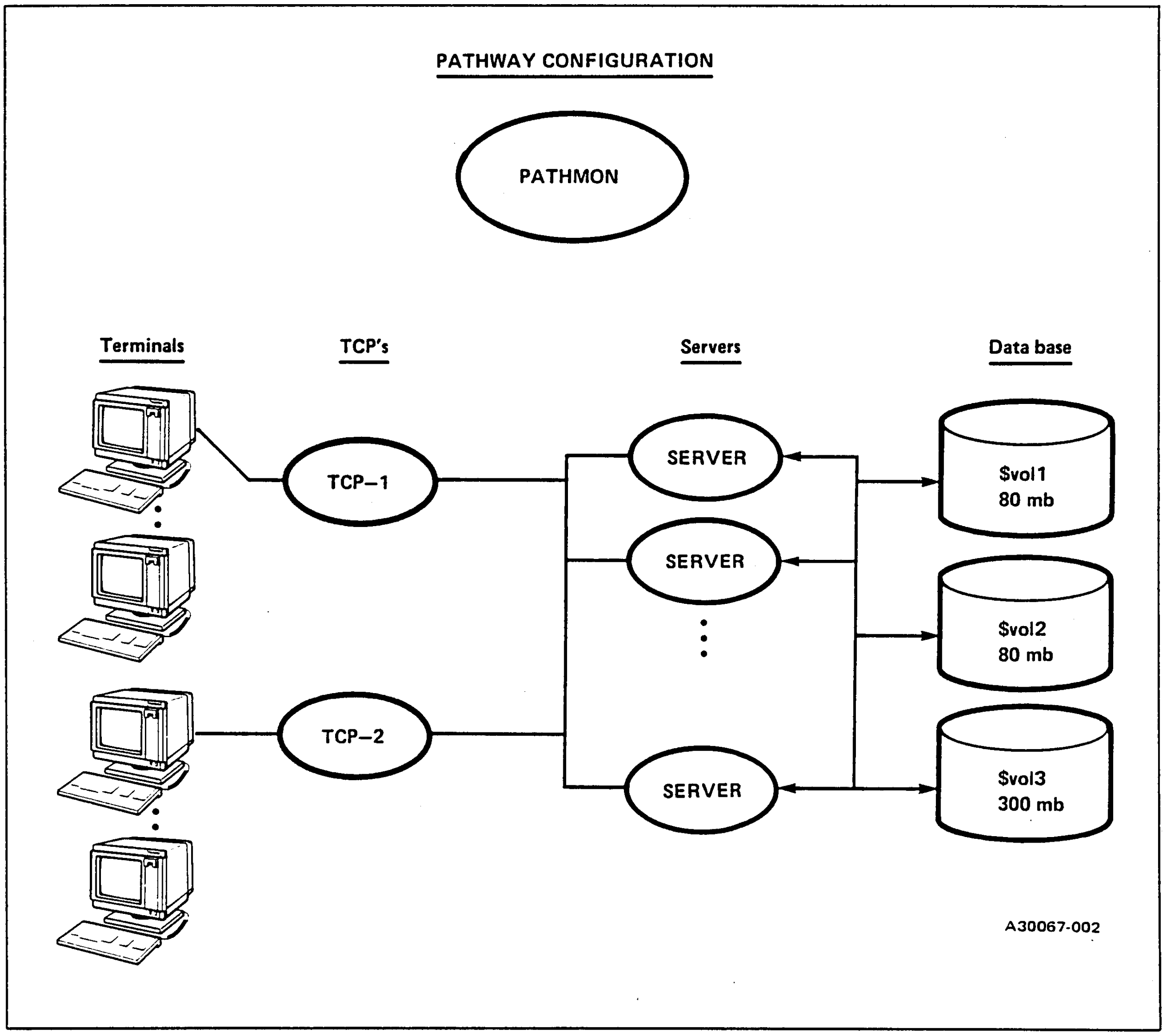


Figure 2. Pathway configuration for EMPACT at a single site.

Application Development

The program development environment consisted of a centralized team of programmer/analysts working with management and manufacturing personnel throughout the company. It was agreed that the development of software to modify the EMPACT database should be controlled by the central team in order to prevent inconsistent alterations to the database. However, all other software, (for example, reports) could be developed anywhere and fall under the control of the intended end user. All released software is archived in a central and controlled repository to protect the integrity of the released software.

Preliminary Designs

At the time the design for EMPACT's distribution was being considered, Tandem had a 50-site corporate network. EMPACT is just one client of this corporate network, which is also used for electronic mail, text processing, program development, and many other EDP applications at Tandem. The network has now grown to 150 sites and is expected to grow accordingly. Each manufacturing site was to have its own system. The question that faced EMPACT's designers was how best to use this network to support multiple sites. The available options were:

1. Remain centralized, have one central database under the control of a single PATHWAY system with remote users accessing the system over the network.
2. Partially distribute the database, have a small number of geographically dispersed *master* sites running independent but identical versions of EMPACT that supported users on sites within a defined region.
3. Fully distribute the database and application; have complete copies of the database and software running at each site.

The first option violated the requirement for continuous availability of the database. Despite Tandem's NonStop™ system design, long-haul communications media are still subject to occasional interruption.

The second option was a compromise solution: vulnerability to significant communications interruptions might, arguably, be reduced, but the principle of site autonomy would not be satisfied. Some individual site or sites would still be dependent on a particular master site. Futhermore, given the geographic dispersion of manufacturing sites, is was unclear whether sites would ever congregate in sufficient numbers to afford nonmaster sites.

These considerations led to the decision to select the third option as the basis for implementing Distributed EMPACT. The next part of this paper describes the specific approach taken.

DISTRIBUTION APPROACH

Data Qualification

The primary components of the design of Distributed EMPACT are (see Figure 3):

- Each site has an independent PATHWAY system running identical applications.
- The database structure for global and local data—as opposed to the actual data—is global, hence identical at each site.
- The global portion of the database is replicated at each site; that is, both the structure and the data of files with global information are identical at all sites.
- The local portion of the database is partitioned over the network; that is, although the structure of files containing local information is identical at all sites, Austin stores the Texas data and Neufahrn stores the Germany data.

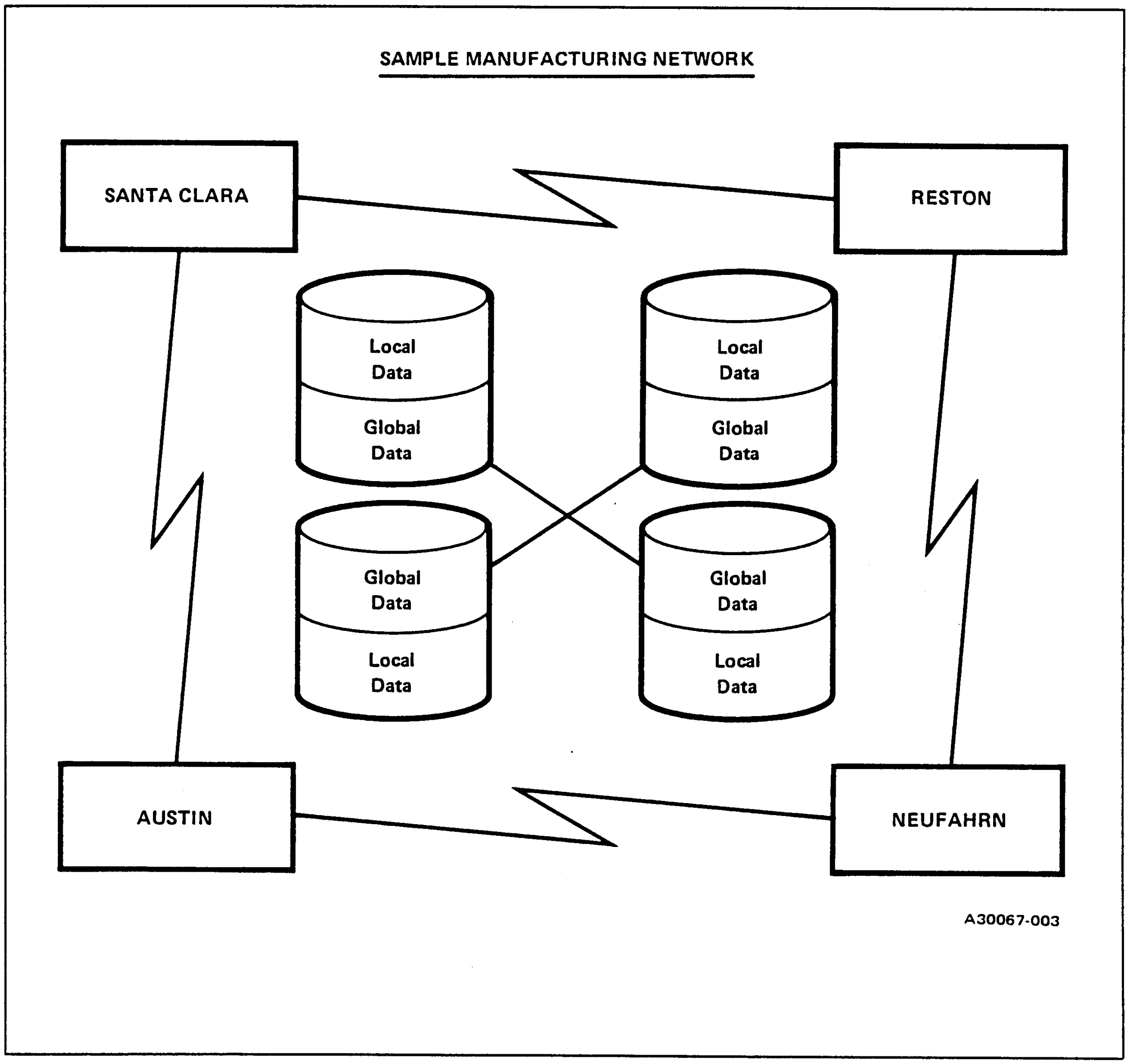


Figure 3. Example of EMPACT network and database structure.

Database Consistency Solutions

This design satisfies some of the objectives of continuous availability and site autonomy. Because a user is concerned only with their portion of the local data and because the global portion is replicated everywhere, query access to the database is guaranteed regardless of the status of other sites in the network or the status of the network itself. However, the issue remains how to provide similar guarantees for update access to the global data yet still maintain consistency among the files.

Maintaining the consistency of replicated files requires that any changes to an individual copy be made to the corresponding copies elsewhere in the network. Furthermore, the changes must preserve the consistency of the files in the event of failure.

Preliminary Broadcast Transaction

One method proposed was to modify the existing servers that updated global files to *broadcast* the updates to all the sites in the network as a single transaction. The transaction would be performed under the auspices of TMF; hence there would be a guarantee, through the *backout* and *rollforward* mechanisms of TMF, that the consistency of the database would be maintained in the event of failure.[1]

This proposal, however, has three drawbacks: servers that perform updates against global files require knowledge of the whereabouts of the file replicas; there is a long response time at the user terminal during an update to global data; and updates to the global files can only take place when all sites are available.

Because the EMPACT sites have a variety of hardware and system configurations, the naming of global files (system name, volume, subvolume) is unique to each site. Because it accesses them directly, the server that performs updates to global files needs to know the names of all global file copies to be updated. Sites in the network have to keep current with the naming changes of all other sites — a problem that becomes considerably worse as the number of EMPACT sites increases.

The terminal response time is unacceptably long because an update to global data involves updating a large number of file copies. On the basis of operator wait-time, daily business requests cannot rely exclusively on this method of update propagation.

The availability of all sites is necessary because by design the broadcast transaction requires access to all affected files at the same time. Data integrity demands an all-or-nothing update. Entrusting the daily business of a site to this method alone sacrifices site autonomy, since legitimate global activity would have to be interrupted until communications were regained. Figure 4 shows the structure of the preliminary broadcast transaction.

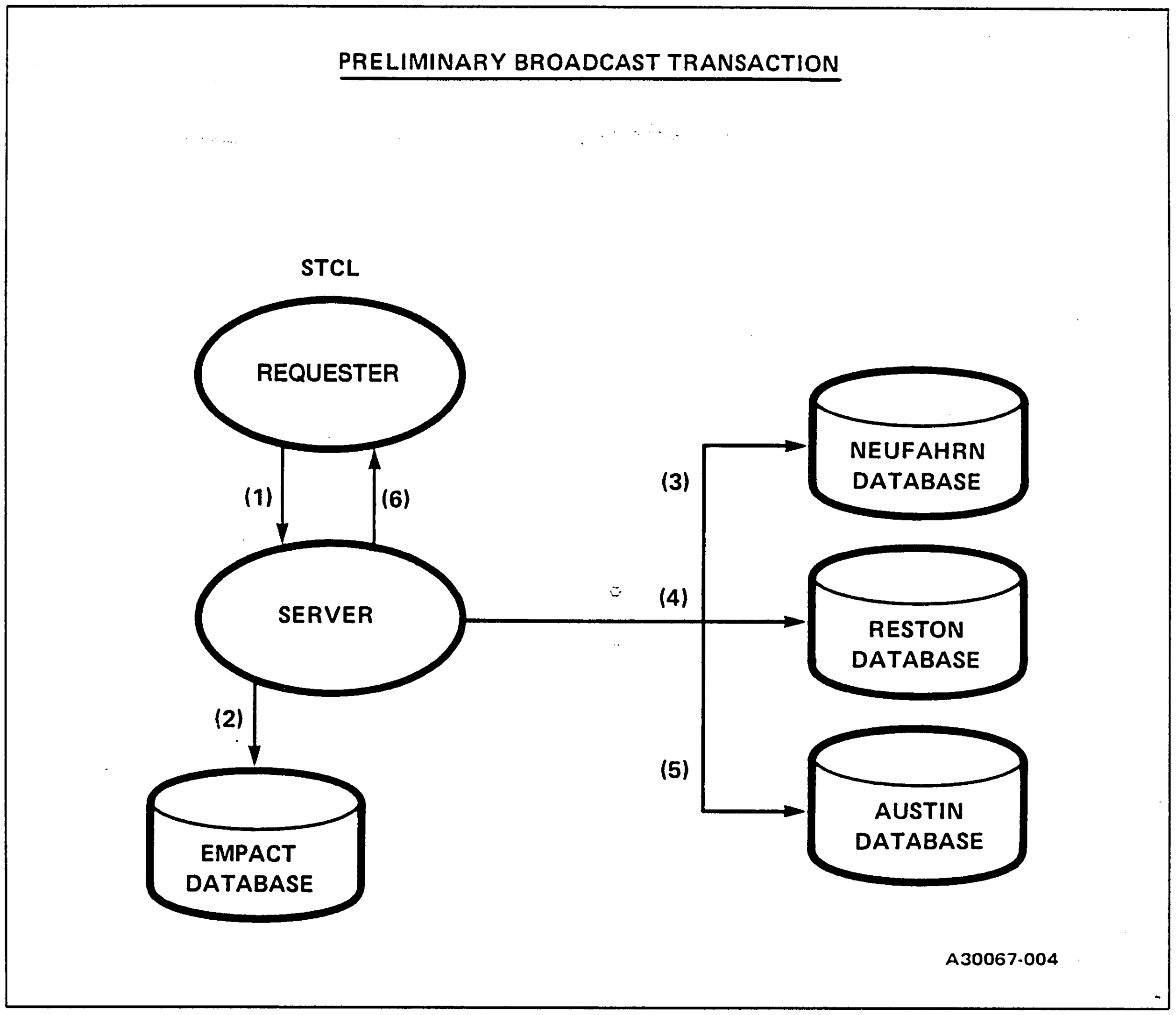


Figure 4. Preliminary Broadcast transaction: (1) Requester sends transaction request message to server, (2) server updates local copy of database, (3)-(5) server updates all known remote databases, (6) server replies to requester.

Broadcast Transaction

A refinement to this proposal defines servers at each site, known as *associate* servers, which, upon receiving a request from an updating server at an initiating site, perform the updates against their copies of the global files. Furthermore, a new global file, known as the node file, contains a list of all EMPACT nodes in the network and thus serves as a network map. A broadcast transaction now involves sending a request to servers at each of the sites listed in the node file. The servers at remote nodes keep track of all physical file placement at their respective sites. TMF guarantees the consistency of the database by covering each broadcast. Figure 5 presents the structure of a broadcast transaction with node file and remote server.

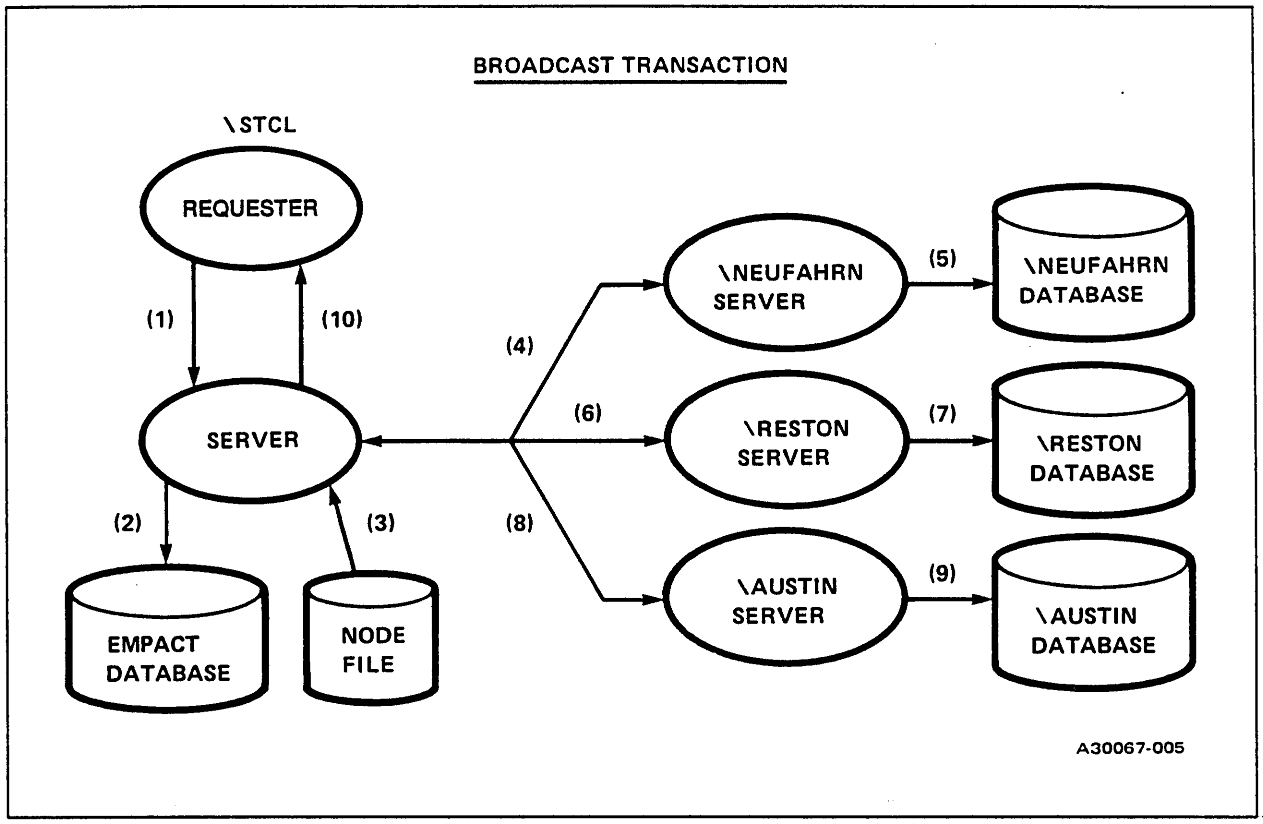


Figure 5. Broadcast transaction: (1) Requester sends request message to server, (2) server updates local copy of database, (3) server reads node file to determine where else to send transaction, (4-6-8) server sends request to associate servers at remote sites, (5-7-9) associate servers perform update, and (10) server replies to requester.

This method of broadcasting transactions resolves the problem of naming; however, it still violates the requirements of site autonomy and short terminal response time. A transaction that updates global data can only be performed when all sites in the network are available. Furthermore, as the number of sites in the network becomes large, the response time on the terminal becomes unreasonably long.

Suspense Transaction

The selected solution to both of these problems was to sacrifice the absolute consistency of the replicated files in exchange for site autonomy and short terminal response times by using a *suspense* mechanism to maintain database consistency.

Instead of immediately broadcasting the transaction to all sites in the network, the server at the site where a global transaction is initiated first updates its copy of the global file and then posts the transaction message to a suspense, or queue, file. A dedicated process, known as the suspense monitor (SUSMON), asynchronously polls the suspense file for transactions and on an as-soon-as-possible (ASAP) basis sends the transaction messages to appropriate servers at remote sites, one at a time, as separate logical TMF transactions. The database is completely consistent only when the suspense files at all the sites are empty. Figure 6 shows the structure of a suspense transaction.

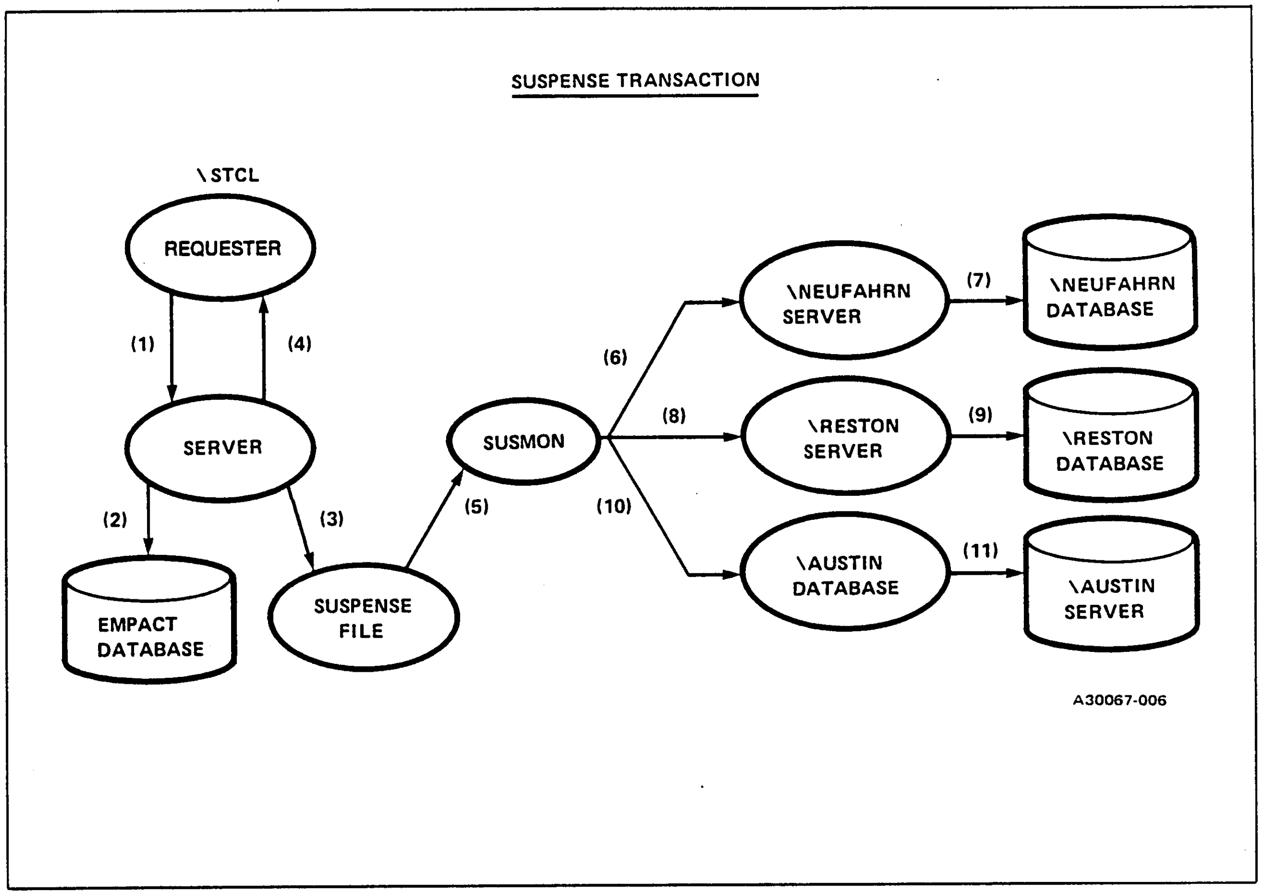


Figure 6. Suspense transaction: (1) Requester sends request message to server, 2) server updates local copy of database, (3) server posts message in suspense file, (4) server replies to requester, (5) SUSMON polls suspense file, (6-8-10) SUSMON asynchronously sends transaction to remote servers, and (7-9-11) remote server attempts update at remote database and respond to SUSMON.

The requirement of site autonomy is satisfied because updates to global files can be initiated regardless of the status of other sites in the network. The problem of unsatisfactory response time at the user's terminal is resolved because the server at the initiating site has only to update the local database before sending a reply back to the TCP. The propagation of the update to remote sites is performed asynchronously by SUSMON. All update and insertion activity involving the suspense file is covered by TMF.

Data Ownership and Key Ranges

As with the broadcast method, TMF guarantees the consistency of the database. However, because the suspense mechanism introduces a time delay in the propagation of updates to remote sites, the possibility of *conflicting* adds and updates among the sites becomes a problem.

Conflicting updates occur when two or more sites update their copies of the same data simultaneously. Similarly, conflicting adds occur when two or more sites add records to a global file with the same key value, but different data. For example, a conflicting add would occur if Reston and Austin were to simultaneously add the same part number to their copies of the item-master file, but have different physical items associated with that number.

To prevent this possibility, ownership (by site) is assigned to global records and the initiation of updates is restricted to only the owning site. To prevent conflicting adds, ranges of key values are preassigned to the various sites and adds are limited to those ranges.

Although only one site at a time can have add or update rights to a particular record or range of records, these rights could be given away to another site. The user community did not find this an unreasonable limitation.

Stale Data

Another problem introduced by the suspense mechanism is the problem of *stale* data. Stale data occur when an out-of-date copy of a global file is read. The data are out of date because an update to the file has been posted at a remote site but has not yet been propagated to the local site. However, because the propagation time for suspense updates is considerably less than the time the user community takes to act on the update, temporary staleness is not a problem.

Before the implementation of distributed EMPACT, the various manufacturing sites were using shared information that was updated and reconciled on a weekly basis. In distributed EMPACT the elapsed time for a transaction to be propagated to all sites in the network is less than a minute; in the worst case, such as a catastrophic network line failure, it is several days. This is viewed as a major improvement. It was also pointed out that, if indeed the most current information were absolutely needed, the read could be directed to the copy at the owner site. If a business emergency were to occur that could not wait on the suspense delay, an outside mechanism such as a phone could be used.

In short, the approach taken to satisfy the simultaneous requirements of site autonomy and database consistency is to replicate communal information, use an ASAP update scheme to maintain the concurrency of the replicated information, and rely on the notion of record ownership to prevent conflicting updates. The favorable relation of propagation time to action time permits brief periods of stale data.

The next Section of this paper discusses some of the details related to the implementation of this approach.

IMPLEMENTATION

The conversion of EMPACT from a centralized application to a distributed application required

1. Specific modifications to the structure of the database.
2. Developing the suspense processing utility SUSMON.
3. Identifying data consistency algorithms.
4. Developing a means to serialize transactions.
5. Deciding upon standard update and record locking protocols.
6. Handling processing variations and exceptions.
7. Error processing.

Database Restructuring

The existing database was reorganized so that the local, global, and semiglobal information reside in distinct sets of files that themselves are solely global, local, or semiglobal. As was mentioned earlier, this restructuring simplifies the organization and reduces the complexity of the update servers. The logic that updates global information can be separated from the relatively simpler logic that updates local information, easing the task of program conversion and maintenance. This conversion is necessary for all programs that access global data.

Suspense File Monitor

The SUSMON process at each site, as shown in Figure 6, is designed to propagate any transaction posted in its suspense file to other sites in the network. Each site has its own suspense file and copy of SUSMON. SUSMON, in conjunction with the servers that update global data, maintains the consistency of replicated data.

SUSMON's map of the network is the node file, which is replicated at all the EMPACT sites. This file contains the GUARDIAN system names and statuses of all sites.

Records in the suspense file contain a copy of the request message, the name of the server class that receives and acts on the message, and a *bit mask* that is matched against the list of sites in the node file to identify where the transaction has been sent or needs to be sent.

SUSMON polls the suspense file looking for work. When finding a transaction that needs to be sent to a remote site, SUSMON performs the following actions:

1. Invokes TMF monitoring—initiates the transaction.
2. Establishes communication with the appropriate server class at the remote site.
3. Sends the request message to the server.
4. Waits for a reply.
5. If the reply is affirmative, updates the bit mask to indicate that the site has processed the transaction successfully.
6. If some site is not available, either because a network link was down or because a system is unavailable, the algorithm defers, intending to try again at a later time.

7. When the bit mask indicates that all necessary sites have received the transaction, the process deletes the record from the suspense file.
8. Ends TMF monitoring—commits the transaction.

Figure 7 diagrams the information flow during suspense processing.

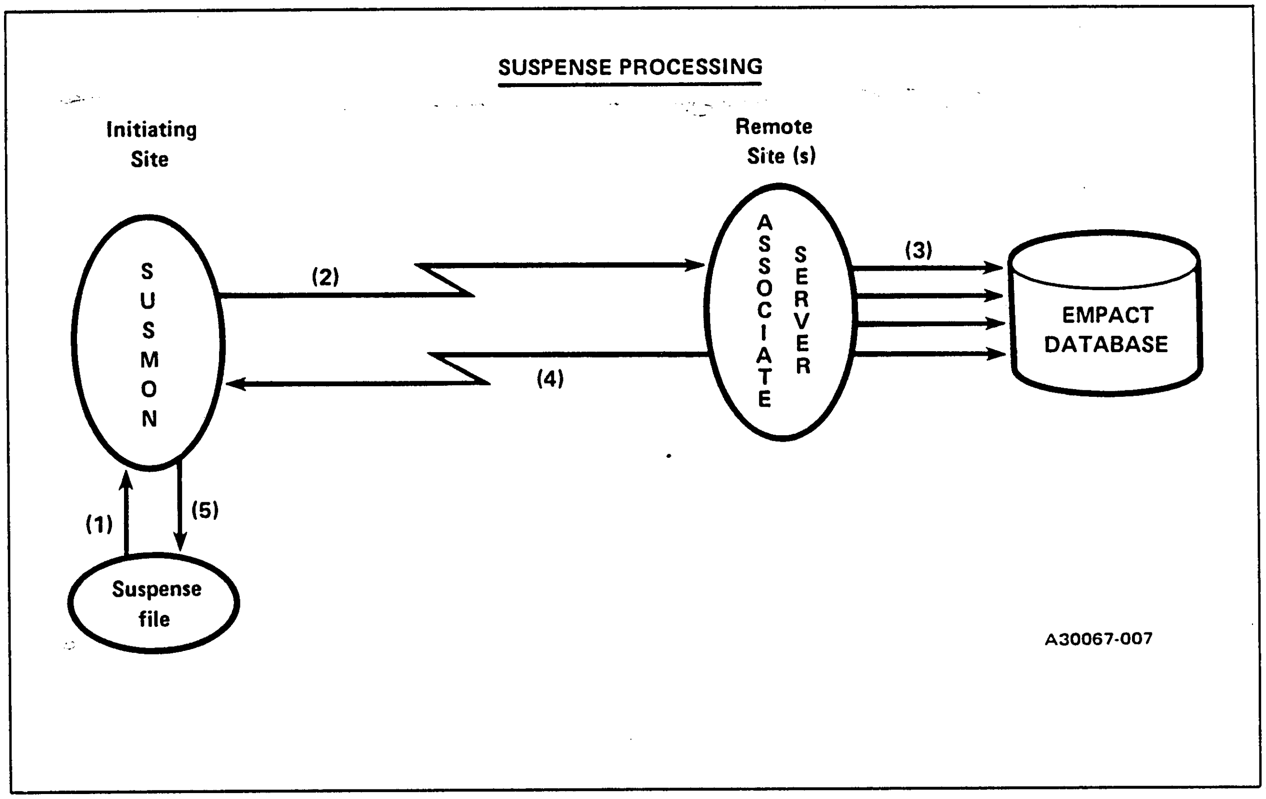


Figure 7. Suspense processing: (1) SUSMON obtains transaction from suspense file, (2) SUSMON establishes communication with server at remote site and sends request message, (3) server updates local copy of database—may involve multiple records, (4) server replies to SUSMON, and (5) SUSMON updates bit mask of record in suspense file.

Data Consistency Algorithms

Given the restructured database and the development of SUSMON, the conversion task was directed towards modifying those servers that update global files. The design of a new update algorithm was required to resolve concurrency problems and to provide a guarantee of data consistency within the replicated files. The original update algorithm, a protocol designed for a single-site application, plays an extended role in the distributed scheme; we will now describe it before moving on to its modifications.

A standard two-step protocol of *check* and *update* is used for update transactions. As part of the first step, an end user might access a screen and make a request to update a certain record or piece of information in order to add a component part to an existing bill of material structure. The requester module servicing the end user then sends the appropriate server class a request for the record or piece of information specified. Next, the server verifies that such an update is possible or permitted, and it replies with the requested data (see Figure 8.)

To add a new component, the server needs to verify, among other things, that the assembly exists, that the component to be added is an existing part, and that the assembly is not a component of the component — bills of materials cannot be recursive.

After the user is satisfied with the data entry, the requester module initiates the second phase of the update process: a request to a server (not necessarily the process that handled the first request) asks that the update be performed against the database. The updating server again verifies that the update is possible, repeating the validity checks that were used in the first step. If all validation is affirmative, the database is updated, a successful reply is sent to the requester, and ultimately to the end user. In the event of error, a negative reply is returned.

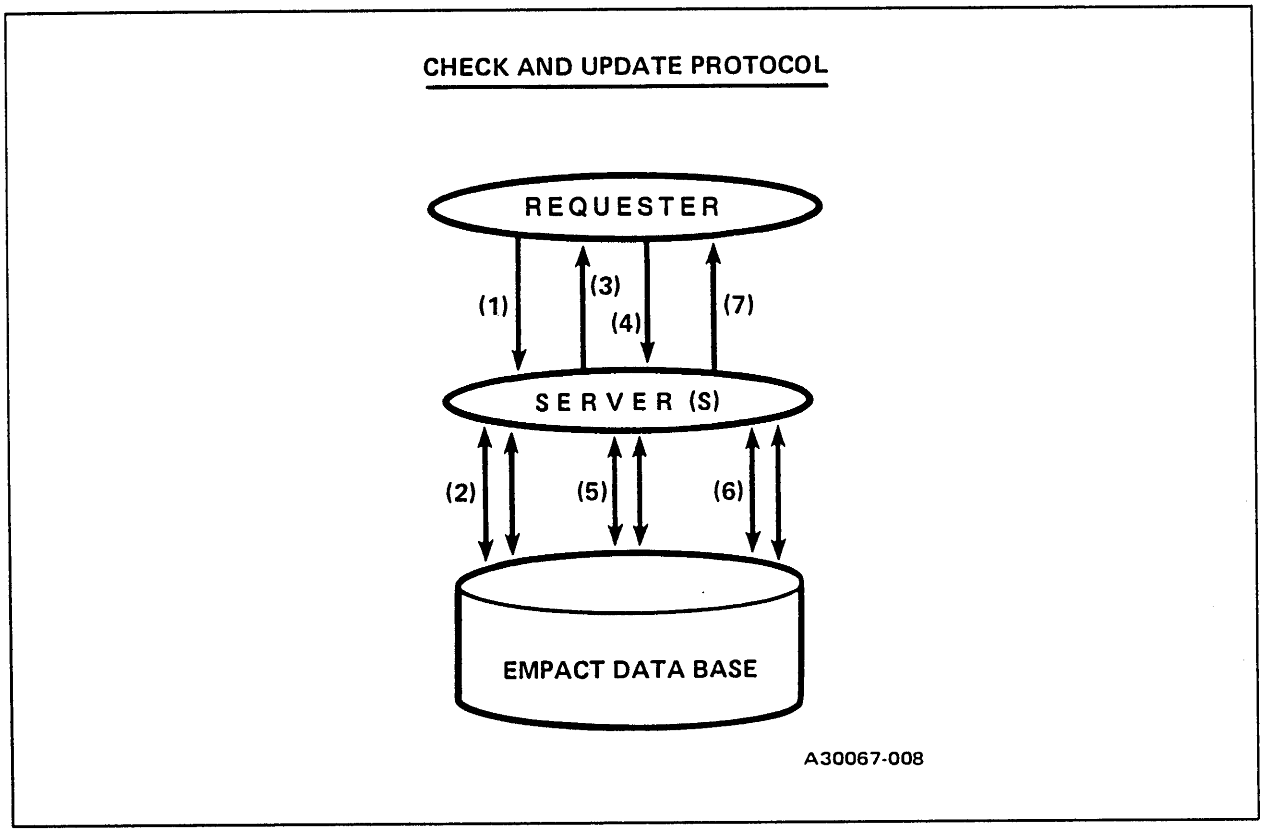


Figure 8. Check and Update protocol: (1) requester sends check message, (2) server accesses database and verifies validity of transaction, (3) server replies, (4) requester initiates update step, (5) server performs validity checks, (6) server updates database, (7) reply is returned to requester. Note that each access to the database steps (2), (5), and (6)—may be multiple.

The purpose of this two-step check and update protocol is to provide a reasonable level of user-friendliness and concurrency. The check allows the user to find out prior to performing potentially unnecessary data entry whether or not a given change is actually needed and possible. A higher level of concurrent processing is achieved because data locks, which are handled at the record level, are only held while the actual update is being performed by the server, and not while data are being entered.

Conflicting updates within a site are prevented because an *old data/new data* comparison is performed before update. If the current copy of the data in the database matches the old copy of the data in the update request message, then the new updated data are written to the database. Otherwise the update request is rejected because a legitimate update has taken place between the time the check request was issued and the time the update request was made.

This two-step method for performing updates still applies within the environment of suspense processing. The check portion of the transaction remains unchanged, but the update portion requires additional logic to make use of SUSMON.

Upon successful completion of an update, instead of simply replying affirmatively to the requester, the updating server first writes a copy of the update request message to the suspense file, then replies. SUSMON asynchronously sends the message, as posted, to identical servers at each of the remote sites. These servers perform the same processing logic that is performed at the initiating site, including validity checks: however, no copy of the message is posted in the local suspense file, since the destination of the update has been reached.

Using identical versions of the update server to process a given request at both the initiating site and the receiving nodes minimizes program maintenance: there is no need for multiple servers with nearly identical logic. Moreover, the consistency of the replicated copies of the data is continually being verified. If a replica becomes inconsistent, this mechanism will detect the circumstance.

This second feature, however, places several additional constraints on the structure of updates to global data. SUSMON sends requests to remote sites serially; that is, requests are sent to remote nodes one at a time and only after the previous one has successfully completed. Because the validity checks are also being performed at remote sites, the success of a global request at a remote node is guaranteed only if the transactions processed at the initiating site are serializable.

Transaction Serialization

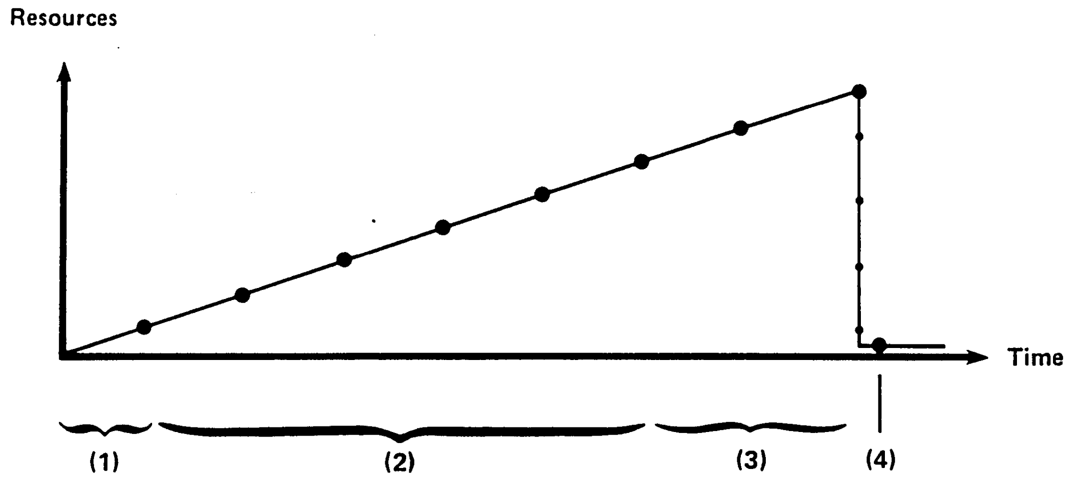
A set of transactions is considered serializable even though it is processed in an interleaved or concurrent fashion, if there exists some serial order that will produce equivalent results.

The method chosen to accomplish this ordering at a node initiating global updates is a counter called the SUSMON-TRANS-ID. The counter is a 32-bit field residing on a single record in the database. Initially the counter is set to zero. Servers processing global transactions are required to read, lock, and increment the counter prior to posting the request message in the suspense file but subsequent to the completion of all standard processing.

Serialization is achieved using the SUSMON-TRANS-ID if

- All records read and written are locked from the time of access to the time that the transaction is committed.
- All records written (added, updated, or deleted) are locked prior to the write operation.
- All records read are global.

This last clause simply implies that there can be no dependencies on non global information, because the condition of non-global data, by definition, may vary from site to site. Figure 9 charts the resource acquisition required for serialization.



(1) TMF transaction initiated.

(2) Reads and writes performed for standard processing, all locks held.

(3) Obtain SUSMON–TRANS–ID and post message in suspense file. (Only one transaction per site is in this step at one time)

(4) Commit transaction and release locks.

A30067-009

Figure 9. Resource acquisition as a function of time during a global transaction.

The suspense file is key-sequenced. The value obtained by incrementing the SUSMON-TRANS-ID counter determines the relative position of the record in the suspense file: this value is used as the primary record key. The order derived is the order in which the transactions will be propagated to other sites.

Record Locking Protocol

The first step of the locking protocol requires that all global records that are read also be locked: this ensures data consistency through the duration of database accesses. The second step of the locking protocol requires that the SUSMON-TRANS-ID be locked by any server process that uses the suspense file; this guarantees serialization of outbound transactions. The TMF Transid, however unique, is not capable of being used to serialize transactions: it is acquired at the time the transaction is initiated, and not at the time the suspense file is written to. Figure 9 clarifies this point.

Variations and Exceptions

The majority of the transactions in EMPACT easily adhere to this structure. There are three notable exceptions, however: the deletion of a part from the item-master, the addition of new global records, and the interplant transfer of material.

Part information in EMPACT resides in the item master record, which is defined to be global. Deleting an individual part from the database involves deleting the associated record from the item master file, but only after inactivity for the part has been established. A part is considered active if inventory records show a stock quantity for the item, or if the part is called out by any assemblies in the bill of materials records. In short, the activity of a part is defined by both global and local information, and it is this dependence on local information that undoes the algorithm outlined above.

The resolution to this problem was the implementation of part deletion as a broadcast transaction. Because the frequency of this transaction is historically low and because the day-to-day business of the individual sites does not depend on their being able to perform this transaction, the fundamental vulnerabilities of broadcast transactions are acceptable. The lesson learned by the designers of EMPACT was that if a particular transaction did not fit the structure required for suspense processing, broadcasting could serve to perform it.

The decision to assign ranges of key values in order to prevent conflicting adds was also the result of an informative debate. Replication requires a common recognition of a record's global uniqueness: for example, a part number can only identify one kind of part throughout the system. If any site were permitted to add a global record by means of a suspense transaction, then the possibility would exist of simultaneous and independent additions of the same record at different sites. On the other hand, if addition of global data were performed through broadcast transactions, then a critical aspect of daily business, the addition of a new part, would depend on the availability of all sites.

The solution reached was the creation of a range file that restricts the add capability of any site to a nonoverlapping set of values. The range file is replicated at all sites and adds can be performed using the suspense mechanism, so long as the key value falls within a valid range. Following our example, each site is permitted to add parts within a unique range of part numbers.

The range file is maintained using broadcast transactions. Paired with suspense processing, the range file concept satisfies all requirements for uniqueness, site autonomy, and user friendliness.

The interplant transfer of material posed a somewhat different problem. As noted earlier, the interplant transfer data are considered semiglobal because the information is relevant only to the sites involved in the exchange of material, which may be two or more sites, but need not be all. The events of a typical interplant transfer of material are as follows: one site requests material from another in the form of a requisition; the supplying site acknowledges the requisition and determines whether there is sufficient available inventory to fulfill the request; if so, the supplying site ships the requested material, and upon receipt the requisition is closed.

A decision was made to implement the interplant transfer functions, or MART subsystem, by replicating the interplant transfer data at the sites involved in the transaction and using the suspense mechanism to move the data from one site to the other.

Suspense transactions in the MART application are composed of potentially multiple requests. The proper processing of these requests at the destination site requires a precise sequencing at the sending site. The internal serialization of a multiple-request transaction can be obtained by incorporating a sequence number into the key of the suspense record: the key is now formed by a counter that contains the value of the SUSMON-TRANS-ID and a sequence number that directs internal order. SUSMON then treats all records associated with a particular counter value as a single transaction but sends the requests, one at a time, to a specified server. The result of these modifications was the evolution of SUSMON from a tool intended solely for broadcasting transactions to a general-purpose transport mechanism.

Figures 10 and 11 describe the differences between a standard SUSMON transaction and a MART SUSMON transaction in terms of the SUSMON-TRANS-ID.

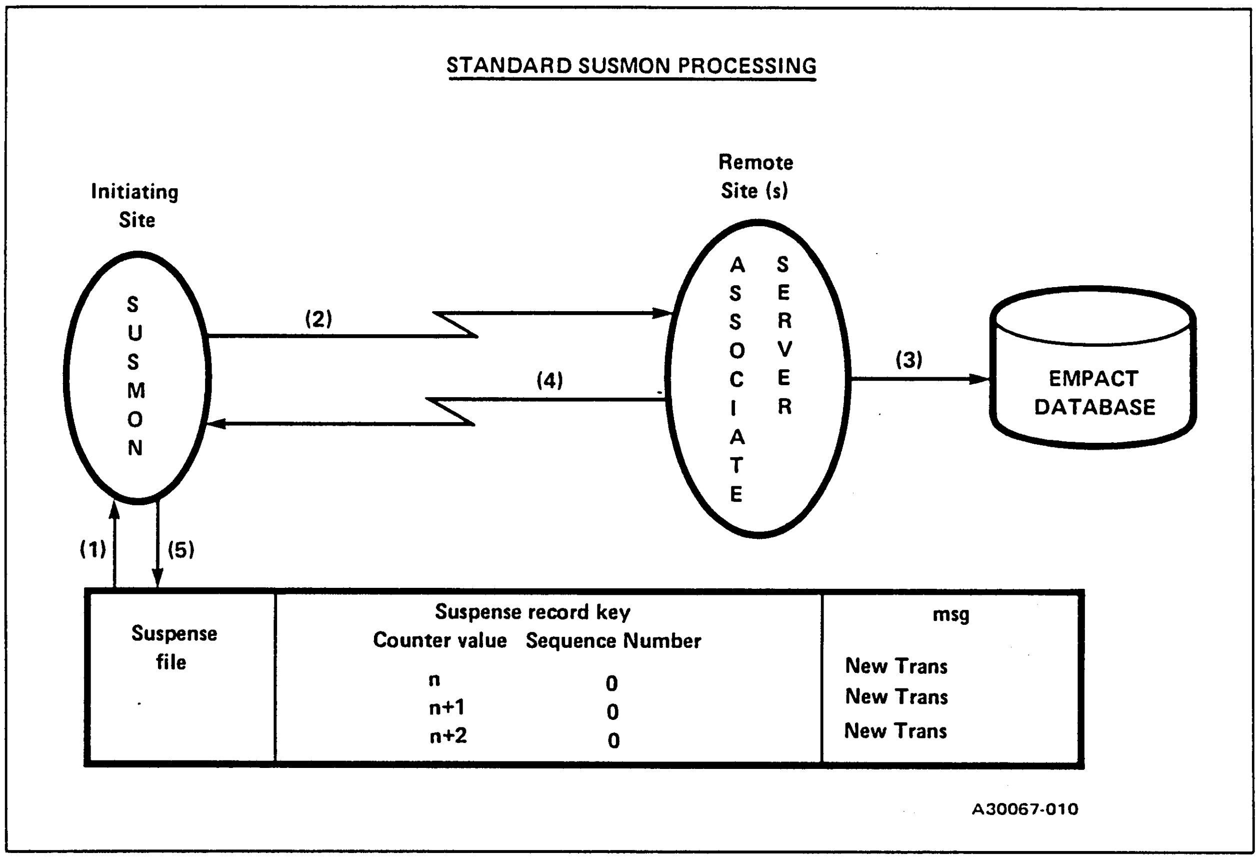


Figure 10. Suspense processing: (1) SUSMON obtains transaction *n0* from the suspense file, (2) SUSMON establishes communication with server at remote site and sends transaction message, (3) server updates local copy of database, (4) server replies to SUSMON, and (5) SUSMON updates the bit mask of *n0* in the suspense file and begins processing transaction *(n + 1)0*.

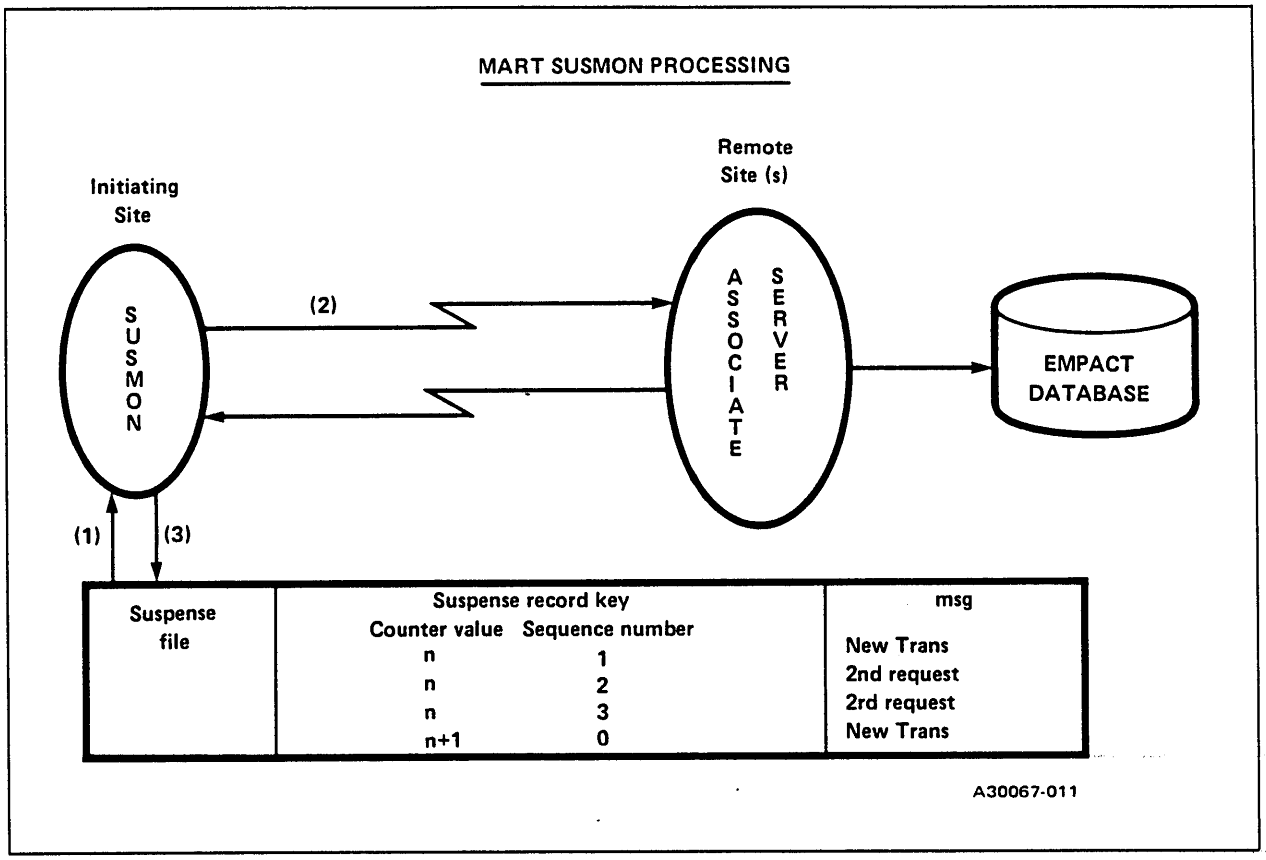


Figure 11. Suspense processing: (1) SUSMON polls the suspense file and observes that the sequence number in the suspense key of the first record identifies the first record of a multi-request transaction. SUSMON recognizes that the unit of work demanded in this case requires that all entries for transaction *n* be processed before completion is reached; (2) SUSMON processes the requests serially in the usual manner; (3) if the server has replied successfully to all requests involving transaction *n*, completion has been reached, the bit mask of *n1* is updated and SUSMON begins processing transaction *(n + 1)0*.

Error determination and handling in a distributed environment requires a level of clarity and coordination that may be unnecessary in a nondistributed setting. A distributed database without a strategy for identifying and resolving errors is singularly vulnerable. SUSMON was chosen as central coordinator of error detection within distributed EMPACT.

Error Processing

SUSMON's interpretation of error conditions determines the subsequent suspense processing to be followed. SUSMON identifies an error as belonging to one of two distinct categories; either the error is retryable or nonretryable. The selection of a category is based upon the following criteria:

1. Retryable—The error was caused by an unanticipated system restriction or problem. Possibilities include communications disruption between SUSMON and a remote server, site unavailability, or isolated system failures within the remote system.

   Solution—The sending SUSMON will report the problem, defer any suspense processing to that site for an ordained amount of time, continue processing to other sites, and eventually retry the transaction.

2. Non-retryable—The error was related to data inconsistency and was reported by the remote server that had attempted to satisfy the request. Possibilities include failure during the remote server's old data/new data check or any other consistency evaluation, or media damage such as an uncorrectable parity error within a record accessed by the remote server.

   Solution—The remote server will report the problem on the remote site, and the sending SUSMON will do likewise locally. SUSMON will indefinitely suspend all activity directed to the site reporting the failure. Resolution of the problem will require a degree of human intervention.

Any problem reporting is done to a hard-copy logging console. It is, moreover, essential that SUSMON suspend all activity to any site that is in error: the serial interdependence of transactions requires the serial processing of any suspense file requests destined to a site. A nonretryable failure of a single transaction to a site effectively interrupts the serial flow of data. Processing to all unaffected sites can continue, since the situation does not imply an error condition outside that reported in the problem site.

EXPERIENCE

The development schedule for distributed EMPACT spans a period of 2 years. The project was commissioned in January 1981 and it was not until September of that same year that the preliminary design was agreed upon. External and internal specifications were completed three months later, and in January 1982 coding began.

The development of SUSMON and the conversion of MART to use SUSMON were completed in April 1982. The conversion of the item master subsystem and bill of materials subsystem, along with the restructuring of the database are scheduled to be finished in December 1982. MART and SUSMON were installed in May 1982 for limited production use. Until the final conversion effort is completed, weekly tapes are being sent to all the sites to maintain concurrent copies of the global information.

The results of the limited production use of MART and SUSMON are encouraging. The software has responded well to the demands placed on on it and the assumptions made during the design phase have proven accurate. Operationally, however, the need for a centrally located network management function has become apparent: events which require the cooperation of all the different sites, such as the addition of a new site to the network, are very difficult to manage manually in a distributed environment.

This need was evidenced during the first attempt to quiesce a three-site EMPACT network. Quiescing a network requires that all servers across the EMPACT network be denied access to their suspense files, and that each SUSMON empty its suspense file of any outbound transactions.

The strategy chosen for this first attempt was to coordinate the activity over the telephone. The exercise demonstrated that the operational complexity of a distributed application demands a high level of training and skill at both the system manager and computer operator positions.

To relieve these difficulties, software was developed to aid in the management of the network, especially in the areas of control and configuration. Its capabilities include

1. The addition and deletion of sites to and from the network.
2. The quiescing of all global transactions.
3. Inquiry into the status of global transaction processing throughout the network.

It should be noted that SUSMON was used to implement the network management software.

CONCLUSIONS

Distributed EMPACT is an example of an application where the requirements of a business call for a distributed solution. The application is considered successful because the structure and organization of the database and software closely parallel the structure and organization of Tandem's business environment.

The design of distributed EMPACT illustrates some of the techniques that can be used in a distributed database application, and the actual implementation of distributed EMPACT demonstrates the feasibility of developing a truly distributed application on the Tandem T/16 system.

ACKNOWLEDGEMENTS

We would like to acknowledge the contributions of Vince Sian and Jim Gray to the initial design of Distributed EMPACT. Further thanks are due to Betty Buhr, Rick Burgess, Bob Plummer, Kathy Welch, and to the entire Manufacturing Information Systems staff for their ideas and advice. We also wish to recognize Carl Goldman and Mike Sanders for the management support that they provided. We are grateful for the help and guidance given us by many members of the Tandem Software Development and Software Education groups, particularly Andrea Borr, Bob Jones, and Jim Collins. Finally, we would like to thank Jim Gray and Rikki Westerschulte for their editorial advice.

REFERENCES

1. Borr, A., "Transaction Monitoring in ENCOMPASS: Reliable Distributed Transaction Processing", Tandem Technical Report TR 81.2, 1981.

2. Katzman, J., and R. Taylor, "GUARDIAN/EXPAND, a Nonstop Network", Tandem Technical Report, 1978.

3. Selinger, P., "Replicated Data". In I. W. Draffan and F. Poole (Ed.), *Distributed Data Bases, An Advanced Course*. Cambridge: Cambridge University Press, 1980, pps 223-231.

4. Smith, L., "Designing a Network-Based Transaction Processing System", SEDS-002, Tandem Computers Incorporated, 1982.