

AD-A082 037

RIA-80-U707

ADA 082 037
TECHNICAL
LIBRARY



RADC-TR-80-7

Final Technical Report

February 1980

DISTRIBUTED DATA BASE TECHNOLOGY STATE-OF-THE-ART REVIEW

Calculon

Calculon

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
MAR 18 1980
S D A

**ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441**

80 3 17 211

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-80-7 has been reviewed and is approved for publication.

APPROVED:

Thomas L. Coluccio

THOMAS L. COLUCCIO
Project Engineer

APPROVED:

H Davis

HOWARD DAVIS
Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER:

John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRDT), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-80-7	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DISTRIBUTED DATA BASE TECHNOLOGY STATE-OF-THE-ART REVIEW		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Mar 78 - Sep 79
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Calculon		8. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0113
9. PERFORMING ORGANIZATION NAME AND ADDRESS Calculon 121 North Broad Street Philadelphia PA 19107		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 45940118
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (IRDT) Griffiss AFB NY 13441		12. REPORT DATE February 1980
		13. NUMBER OF PAGES 79
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADC Project Engineer: Thomas L. Coluccio (IRDT)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Distributed Data Bases Data Base Integrity Information Systems Distributed Processing Data Base Security		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This document provides theoretical information relating to the development of information systems where large data bases are utilized. The study considers the problems associated with the construction, implementation and maintenance of those data bases in a distributed processing environment. The data base management systems, data communications networking and the utilization of multi-processor configurations are also included in the investigation. Four specific problem areas of the update function under the distributed concept (Cont'd)		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Item 20 (Cont'd)

were studied and are reported upon. They are data integrity, logging and recovery, deadlock and security.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Table of Contents

Paragraph	Title	Page
	<u>SUMMARY</u>	iii
	<u>I. INTRODUCTION</u>	1-1
	<u>II. DATA BASE INTEGRITY</u>	2-1
2.1	INTRODUCTION	2-1
2.1.1	Objective.	2-1
2.1.2	Distributed Data Base Systems as the Context for Integrity .	2-2
2.1.3	Definition of Data Base Integrity.	2-2
2.2	DEFINITION OF THE PROBLEM.	2-3
2.2.1	Component Outage Threat.	2-3
2.2.2	Concurrent Update Threat	2-4
2.3	SOLUTIONS TO DATA BASE INTEGRITY PROBLEMS.	2-5
2.3.1	Integrity Loss Due to Outages.	2-5
2.3.2	Integrity Loss Due to Concurrent Updates	2-6
2.4	ASSESSMENT OF THE STATE-OF-THE-ART	2-10
2.5	AREAS FOR FURTHER STUDY.	2-11
	<u>III. LOGGING AND RECOVERY</u>	3-1
3.1	DEFINITION OF THE PROBLEM.	3-1
3.1.1	Logging and Recovery Functions	3-3
3.1.2	Logging and Recovery Operations.	3-4
3.2	DISCUSSION OF SOLUTIONS.	3-7
3.2.1	Logging.	3-7
3.2.2	Rollback and Restart	3-9
3.2.3	Recovery and Restart	3-10
3.3	ASSESSMENT OF THE STATE-OF-THE-ART	3-15
3.4	AREAS FOR FURTHER STUDY.	3-16

Table of Contents (continued)

Paragraph	Title	Page
	<u>IV. DEADLOCK</u>	4-1
4.1	INTRODUCTION	4-1
4.2	THE DEADLOCK PROBLEM	4-2
4.2.1	Concurrency Control.	4-2
4.2.2	Locking.	4-3
4.2.3	Occurrence of Deadlock	4-4
4.2.4	Deadlock in Distributed Data Base Systems.	4-4
4.2.5	Approaches to Solutions.	4-5
4.3	SOLUTIONS TO THE DEADLOCK PROBLEM.	4-5
4.3.1	Deadlock Prevention.	4-7
4.3.2	Deadlock Avoidance	4-14
4.3.3	Deadlock Detection and Resolution.	4-15
4.4	ASSESSMENT OF THE STATE-OF-THE-ART	4-18
4.5	AREAS FOR FURTHER STUDY.	4-18
	<u>V. DATA BASE SECURITY</u>	5-1
5.1	THE GENERAL SECURITY PROBLEM	5-2
5.2	RELEVANT SECURITY TECHNIQUES	5-3
5.2.1	Telecommunications	5-4
5.2.2	Operating System	5-6
5.2.3	Data Base Management Systems	5-7
5.3	ASSESSMENT OF CURRENT TECHNOLOGY	5-8
5.4	AREAS FOR FURTHER STUDY.	5-9

SUMMARY

For several years, RADC has participated in the development of information systems to assist analysts in the construction, maintenance, and utilization of large Scientific and Technical (S&T) data bases. Current trends in computer technology development are leading system designers to consider distributed processing concepts and, in particular, distributed data bases. These trends are exemplified by the implementation of more comprehensive data base management systems (DBMSs), far-flung data communications networks, and multiprocessor computer configurations.

In present and projected S&T environments, system designers face complex trade-off evaluations. The benefits of distributed data base systems are appealing, but they are not available without cost. The costs of additional processor, storage, and communications hardware are readily identifiable; costs of increased complexity or reduced performance are less easily determined.

To aid in the evaluation of design alternatives in a distributed data base environment, a study of certain specific problem areas has been undertaken. The study focused on the updating of distributed data bases. Updating can involve portions of a data base that are replicated at multiple nodes in a network, and can involve transactions whose entry, processing, and data modification each take place at a different node. Several update transactions can be simultaneously extant in such an environment, and enormous processing and control complexity can result.

Four specific update problem areas were studied, viz:

- Data integrity
- Logging and recovery
- Deadlock
- Security.

A review of the literature was conducted to ascertain the present state-of-the-art for each problem area. Each was defined as it occurs in distributed data base systems. Solutions proposed in the literature were

described, compared, and evaluated. The state-of-the-art was assessed, and areas were identified where further research is needed.

It was found that the problems are not unique to distributed data bases; they exist also in other, multiuser systems. Distributed data base configurations, however, compound the problems manyfold. While, in general, the same solution alternatives prevail as in simpler systems, their implementation in this environment would be much more difficult and costly.

Most of the reported work is theoretical; few of the proposals have been implemented. Many of the concepts have not been well developed for distributed data bases; the references frequently state in effect that the situation is more complex in distributed data base environments, and they then return to discussions of simpler systems. Virtually no quantitative data is reported.

The state-of-the-art is judged to be immature, with considerable further investigation called for. Of the four areas addressed, security is probably the least developed. The problem areas studied here are only part of the overall problem of designing usable distributed data base systems, and continued parallel study of distributed data processing, DBMSs, communications networks, and distributed data bases is recommended.

The study report contains an introduction and individual papers on the four subject areas. Each paper includes a bibliography.

EVALUATION

This effort was designed to investigate the problems associated with the development of distributed processing systems, in general, with specific emphasis on the problems of the update function. The effort was undertaken under the RIE (Intelligence Integrated Systems) Technical Planning Objective and coincides with other efforts within that thrust.

The value of the effort, although mainly qualitative, rests with the information contained herein that can be applied by a systems designer of a distributed system when faced with the trade-off decisions. The effort also reveals the direction in which future research of this nature should be aimed.


THOMAS L. COLUCCIO
Project Engineer

I. INTRODUCTION

Distributed data base processing is a state-of-the-art computer capability. It continues to stimulate many related new systems ideas, techniques, and architectures.

Distributed processing, in essence, is the utilization of multiple connected computing facilities to handle data processing requirements. Various distributed processing hardware networks can be defined, including a centralized hierarchical network, a centralized/decentralized star network, or a decentralized ring network. Associated with the hardware, various distributed data base organizations can be defined, ranging from a single central shared data base, to local data bases with some central control, to totally local data bases.

The major aim of distributed processing is to place the processing power and data base where it is needed, even if multiple copies of the data base are required.

The field of distributed data base processing, being so young, is not yet based on a foundation of well-developed and generally accepted concepts and assumptions. One area in which there is general agreement among the research

and development (R&D) community is general system architecture of a distributed system; namely, a collection of independent computers, each of which supports a portion of the overall data base. These computers would be interconnected via some type of communication channels which are used to transfer data and control information. Users would be permitted to enter transactions at any of the computers; the users' transactions may reference data stored at a remote site, in which case the distributed system accesses the data by means of the communication channel. In some cases, the problem of finding remote data would be automatically handled by the system; in others the user must explicitly tell the distributed systems the site at which each datum is stored.

The formation of a distributed data base system across computing facilities increases the control software complexity. This complexity depends on the processors, storage devices, and data base management systems employed, as well as on the level of data base control desired. The simplest system is one that employs compatible processors, storage devices, and data base management systems; the most complex system involves different processors, different storage devices, and different data base management systems.

Distributed Data Base Considerations

Within general distributed system architectures, many different approaches can be taken regarding distribution of data. For example:

- (1) Each data base site could contain a complete copy of the entire data base.
- (2) Each data base site could contain a unique subset of the data base which:
 - Does not overlap the portions stored elsewhere; this partitioning of the data base could be accomplished by assigning each file to a single site, or the unit of partitioning could be smaller.
 - Does overlap the portions stored elsewhere; the impact of this generalization is that it allows redundant data to be stored in the data base, meaning that the same logical data item is physically stored at several sites. In such cases, the system is responsible for automatically updating all copies of the redundant data in response to users' update transactions.
- (3) There may be various combinations of copied and unique subsets.

Most current R&D operations favor some form of partially redundant (replicated) data. The distribution of a data base over a computer distributed network enormously complicates the data base administration function. Perhaps the most important factors are the distribution of the data across machines and the amount of data integration. As distribution and integration of data increase, the data base becomes more accessible to a larger number of users. At the same time, control operations become increasingly complex. This quandry reduces to the classical data processing trade-off between flexibility and efficiency. It is the role of the system designer to balance these two seemingly conflicting factors.

The objective of this technical report is to present a survey of the distributed data base state-of-the-art with respect to handling specific problem areas in the design of distributed data base computer systems. This report is intended to be a tutorial for the use of Air Force program personnel and contractors, to guide them in the design of Indications and Warning (I&W) distributed processing intelligence networks. It is heavily based upon material found in professional literature. Although this report focuses on computer systems containing distributed data bases, concepts applicable to computer systems in general, or to distributed processing networks with centralized data bases, are not excluded, but it is their relevance to distributed data base systems that is of interest.

Within the area of distributed data base systems, four specific major areas were studied; all four are related to updating within a distributed data base environment. The four areas, discussed in the following four sections, are:

- Data Integrity
- Logging and Recovery
- Deadlock
- Security.

Each section contains:

- Definition of the problem
- Discussion of solutions
- Assessment of the state-of-the-art
- Areas for further study.

Each section also contains a bibliography of applicable source documents. In order to facilitate independent use of the four topical discussions, some repetitive descriptions appear among the four ensuing sections.

II. DATA BASE INTEGRITY

This section discusses the problem of maintaining the integrity of a distributed data base. Integrity is here concerned with correctness and consistency.

2.1 INTRODUCTION

In the distributed data base systems literature the subject of data integrity figures prominently, particularly as regards the impact of the update functions. This section of this investigation covers the data integrity issue including those activities which adversely affect data integrity, together with the solutions described in the literature, and brief comments are offered regarding possible future activities in data base integrity.

2.1.1 Objective

As organizations move towards the use of shared data bases, it becomes increasingly important to be able to maintain the integrity of shared data resources. This is because those managers and users who previously were in control of the data they used are now being asked to give their data resources

to a data base administrator over whom they have no control, while they retain the responsibility for results based on that data. The willingness of managers and users to release their needed data resources and to trust someone else to maintain the integrity of their data base will depend, at a minimum, on the establishment of a strong procedure for the maintenance of the integrity of the data bases. This discussion is intended to comment on certain aspects of this concern by users.

2.1.2 Distributed Data Base Systems as the Context for Integrity

Data base integrity is a concern for both centralized and distributed data base systems. In a central data base system, all the data base and its data management functions are concentrated in a single processing site, even though there may be other processing functions at other locations. With close central control of the data, and controlled access to it for inquiry and update, the data integrity functions are simplified, and consequently relatively easily managed. However, once the decision is made to distribute the data base among a number of nodes (or sites), then an additional level of complexity is reached, particularly when multiple copies of files are located at the various nodes. Because the distributed data base environment is the subject of this investigation, and because it represents the more complex and general situation, only the distributed file arrangement will be considered in this discussion of threats to integrity and potential solutions.

2.1.3 Definition of Data Base Integrity

In the broadest sense, data base integrity implies the correctness and consistency of the stored data. Additional integrity-related issues such as security and confidentiality of the data are outside the scope of this discussion.

- (1) Correctness. In a condition of integrity, the stored data should be correct, in the sense that the data remains that which was placed in storage. The data base system is not responsible for the absolute accuracy of all facts, since wrong data could have been entered, but it is responsible that the data should be changed only when intended.

- (2) Consistency. In a distributed data base system, information generally exists in multiple or redundant copies at the various nodes. For data base integrity, it is essential that the multiple copies of each file are consistent and no differences exist among the several copies of the same data. This applies both where files are duplicated and where the same facts are held in different files. Where the same facts are stored, the values should be consistent.

In a distributed data base system, the key integrity issue is that many processes are going on at once at various locations using the various copies of the data base. Without appropriate integrity safeguards the hazard is that the various copies of the files will be modified differently and as a result the integrity of the data base will be compromised.

2.2 DEFINITION OF THE PROBLEM

This chapter discusses the threats to data base integrity which are characteristic of distributed data base systems. (Other threats which are outside the scope of the investigation or are discussed elsewhere are physical threats to the system and data base security issues.) The key threats to data base integrity of interest result from outages and from the fact that at least portions of the data base are replicated at more than one node. As a result of either loss of access to a copy of a file or to concurrent inquiries and updates to a file, copies of files can get out of synchronism resulting in loss of data base integrity.

2.2.1 Component Outage Threat

The most obvious threat to integrity results from a component outage in the distributed data base system. Such a component outage could be a processor, a storage module, or a communication link that prevents the system from keeping a part of the data base from being kept in synchronization with its counterparts elsewhere in the system. If a node is cut off from the system by communications failure, it could continue to provide service to its connected users, employing its local data resources. However, any file updates made either locally or elsewhere in the system will cause those files (and any later transactions based on those files) to become unsynchronized or inconsistent.

This will not only lead to erroneous results but will also complicate the process of bringing the data bases back into coincidence when service is restored. This will not only lead to operational problems (data and decision errors) but it may also have legal implications.

2.2.2 Concurrent Update Threat

Whenever a process updates a data base concurrently with another update process, the integrity of the data base is threatened. Furthermore, a reading (inquiry) process is threatened by an update process, which can give the appearance of a loss of data integrity. Rigid control of data base access can be obtained in a central data base system but in a distributed data base system, with numbers of copies of the data base and many processors in operation, the concurrent access problem is less easily controlled.

For example, given two independent, concurrent processes, P and Q, which attempt to update the same record A at the same time. The following events may take place:

- Process P requests and receives a copy of record A
- Process Q requests and receives a copy of record A
- Process P modifies its copy of record A and writes it back in the data base
- Process Q modifies its copy of record A and writes it back in the data base.

Because process Q has carried out its processing and update of record A, the update action of process P has been lost and, as a result, the integrity of record A has been lost. This resulted from the independent action of the processes. Although this example is for a single data base and two sequential processes, the effect is the same when the actions take place in separate processors using different copies of the same file.

A corresponding effect can take place if one process R is, for example, posting transactions to the file while another process, S, is preparing a trial balance. Depending on the timing of the processes, it is possible that the

debit side of a transaction will be posted in time to be included in the trial balance, while the corresponding credit side of the transaction will not be posted in time to be included, so the trial balance would be out of balance. In this case, although the data base integrity was maintained in fact, the integrity of the process (on the inquiry side) was compromised and the data base integrity will appear to have been lost.

2.3 SOLUTIONS TO DATA BASE INTEGRITY PROBLEMS

Review of the literature relevant to distributed data base integrity has not revealed any generally satisfactory solution to the integrity problem. The use of a lockout mechanism is described as an obvious approach which leads to the possibility of deadlock, discussed elsewhere. The lockout approach, however, is a scheme which effectively forces the distributed system to function as a central data base system under central control, on a transaction by transaction basis.

This chapter will comment further on identifying when loss of integrity has occurred, on lockout, and on other approaches to ensuring data base integrity.

2.3.1 Integrity Loss Due to Outages

The most obvious integrity threat in a distributed data base system is that due to outage, particularly of a communication channel between nodes. If a node is isolated from the system due to communications loss, and it continues to service its users to the extent possible, then its data base can get out of synchronism with the data base in the remaining part of the system.

This integrity threat can be countered by arbitrarily cutting off service to users at the isolated node, so that transactions cannot be made against its copy of the data base. Because some fraction of the transactions taking place at a node is likely to require communications with parts of the data base located elsewhere, this arbitrary cutoff in service is not as serious as it might at first seem. With this approach, the users know exactly where they stand, rather than being uncertain as to whether a transaction will be completed, or whether they will later have to back it out during the recovery process.

Another approach would be to limit transactions at the isolated node to file inquiries. Although this might at first seem useful, it could result in users taking action based on data that later turns out to have been changed during the outage period. In addition, even in an inquiry-only situation, it is likely that a significant fraction of the desired inquiries would require reference to other nodes. As a result, not all inquiry requests could be processed, and the user would not generally know whether he would receive service or not. Consequently, it is recommended that no attempt be made to respond to data base inquiries at the isolated node.

Under conditions of communication loss the isolated node can, however, provide other services to users, which do not require access to the shared data base. For example, the processor would presumably inform user terminals of a cutoff of specific services, and the processor could be used to support users by queuing up requests for service, so that the users need not keep manual records to be input when network service is restored. Upon restoration of network access, the first task for the processor (and for the system) would be to bring the local data base up to date with the system. This would be done for those files which had been changed during the service outage, and no transactions would be permitted during the recovery process. The users will remain locked out until after the recovery process is complete, when the pending transactions could be released.

2.3.2 Integrity Loss Due to Concurrent Updates

The integrity threat due to concurrent update is less obvious and more troublesome. As noted in paragraph 2.2.2, a series of independent updates of a file can result in an update being lost, and independent updates and inquiries can result in action being taken on data that is no longer current. The general solution to this problem appears obvious; to lock out the files from being accessed by more than one process at a time.

Lockout is a process of mutual exclusion in which the object file (or other subset of the data base) of an update action is accessible to only one process at a time. Lockout is straightforward to implement in a single data base system. In a system with multiple data bases or multiple copies of

the data base, the use of lockout creates a significant processing and communications load, which grows rapidly with the number of copies of the data base and nodes, and which adds substantially to transaction turnaround time.

Lockout creates a processing and communications workload because, before initiating a transaction against the file, the required resources to carry out the transaction must all be exclusively allocated to the process. This requires a series of messages to each node holding the file before, during, and after the transaction. These communications, and the resulting processor/file actions, introduce a lengthy delay in executing each transaction, which can become two to three orders of magnitude greater than the lockout process in a centralized data base system.

For example, a straightforward lockout algorithm in a distributed system of N nodes requires $5(N-1)$ intercomputer messages, in addition to the processing required locally in the node initiating the transaction. Messages must flow between the initiating node and each other node as follows:

- Lock Request Message
- Lock Grant Message
- Update Message
- Update Acknowledgment
- Lock Release Message.

In addition to each message delay, the update execution is also lengthy. Because of these delays, a straightforward lockout approach is considered by some to be unsatisfactory in a general-purpose distributed data base system, and other methods have been sought. Some of these approaches seek more efficient ways of propagating the lockout information through the network. An additional consideration is the hazard that all sites or communication channels may not be operational when the process begins (or that a site or channel will become inoperative during the process).

Another approach is for the node where the transaction originates to forward the transaction to each of the other nodes. Each node then carries out the update process independently, and the affected file is locked out from other transactions for a predefined period. This approach reduces the message traffic between nodes, but significantly increases the hazards, due to the possibility

of communication line outage, or processors or files busy. The process is open loop and there are no confirmation messages from the nodes. Therefore, there is the possibility of the update not being carried out in each node, and, a copy of the data base getting out of synchronism. There is also the problem that the required lockout time at each node to complete the update will depend on local traffic, seek time, etc., which will be unknown to the initiating node. Thus, a sufficiently long fixed lockout time will, on the average, increase the time associated with carrying out each transaction. Also, there still is no assurance that all nodes will be updated for every transaction, since there is no clear way to handle the contingency that a communication line, node, or storage unit is out-of-service. Thus, the blind, open-ended forwarding of transactions to nodes for action, without confirmation or followup does not appear to be a desirable approach to consider.

In the next approach to lockout, designed to reduce the number of messages between the originating node and other nodes, the update is first carried out in the originating node. If successful locally, it is presumed that the update will be successful at the remote locations. The lockout and update messages are then transmitted to the remote nodes and executed, and then update acknowledgments are received back from the nodes. When received, the originator can then transmit lock release messages to the other nodes. Thus, by using a two-stage process, the number of messages can be reduced to either $4(N-1)$ or $3(N-1)$ messages, depending on whether the update transmission is combined with the lock request messages or not. A key problem with this approach that has not received adequate attention is, as with other approaches, what happens when nodes, communication channels, or storage modules are out of service, i.e., where are the transactions stored, and how are they reactivated when equipment comes back on-line. Beyond this, consideration must be given to the additional complexities which result when a component goes out of service during any stage of the process.

An algorithm that further reduces the message traffic to $2N$ messages has been proposed in the literature. In this case, messages are transmitted sequentially (daisy-chained) from the originating node to the next node, and then from that node to the next, and so on. While this reduces the message

volume substantially, it also greatly increases the communications delay time. This approach also gets into difficulty if a node is out of service, because the message chain is broken. Consequently, without a sophisticated communications fallback scheme, this approach is unlikely to be practical in a live environment.

Communications volume may be further reduced to about $1.5N$ by a modified daisy-chain approach, with a voting protocol for lockout setting. However, it also introduces communications delays and, without a workable fallback/outage protection scheme, it is also likely to be unacceptable in practical situations.

Other approaches seek to reduce synchronization costs by reverting to a form of centralized control of the distributed data base. It has been proposed that one site be designated as a primary or controlling node for a specific file or set of files. Thus all update activity for a given file would be funnelled through its controlling node, regardless of where it originated. This approach appears attractive for simple applications where it is possible to partition data base activity on a geographical basis. However, in a more complex situation the primary site approach cannot avoid the need for global data base lockout, and the communication costs, delay, and complexity become high (possibly higher than other approaches due to the need for communications with the primary node). This approach also results in increased complexity to handle the consequences of primary site failure (primary site control functions must be transferred to another location, etc.).

Reviewing these various proposed approaches to handling the concurrent update problem it appears that, in spite of the communication costs and delays, and the problems that can result from out-of-service components, the straightforward lockout arrangement is to be preferred. Lockout schemes introduce the possibility of deadlock, which is treated in Section IV. Given solutions to the deadlock problem, methods are needed for improving message-exchange between nodes in order to increase communications efficiency and reduce transaction delay. This is particularly important as the number of nodes becomes greater.

Our review of the literature leaves two general impressions. First, much of the published work leans toward the academic or theoretical without hard solutions to specific problems. Integrity is treated as a requirement or a goal but the means of achieving that goal are generally not described. Second, data base integrity is said to be principally threatened by the problem of concurrent updates, the solution to which is lockout, or control of access by the system to the various copies of the data base. The consequences of lockout are both increased communications traffic and service time, and the threat of deadlock. Alternative ways of reducing the communications and processing loads associated with lockout appear to be a main concern of the literature.

What does not appear to be published are approaches that discuss practical aspects of assuring data base integrity. Information is not presented on specific examples of lockout used in operational systems, together with information on the performance of these examples. Although such information for specific systems may be considered proprietary, it is also possible that little firm information exists. If available, specific performance information for various approaches would be useful to the system designer.

Considerable study has been devoted to the problem of update synchronization, to maintain data base integrity. In practical distributed data base systems real time update of multiple copies of the data with varying delays are the key threat (concurrent updates) to integrity. Lockout mechanisms with serialized updates applied in order to all copies of the data base are required to ensure integrity. The various update/lockout mechanisms mentioned in the literature are designed to minimize delay and communication cost, avoid preferential treatment of nodes, and minimize the need to back out updates. The design of the system to assure distributed data base integrity must, however, be evaluated in the context of the application. It is clear that there is no unique or optimum solution to the integrity problem, and that integrity assurance must be an integral part of the system design process.

As noted above, the straightforward lockout approach, even with the deadlock problem and the required message traffic between nodes, appears to be the cleanest method for dealing with the threat to distributed data base integrity due to concurrent transactions. However, practical methods are certainly needed for improving communications efficiency, perhaps by a pragmatic, rational analysis of which nodes are involved in each transaction (rather than assuming that all are). Also needed are practical procedures for dealing with failures, particularly those occurring during the lockout process.

Review of the published literature gives the impression that much of the development work in distributed data base integrity leans to the theoretical or academic. Developers of distributed data base systems in the real world, however, must solve these hypothetical problems in a realistic, practical way. For example, it should be possible to assess the probability of data base integrity loss due to concurrent transactions based on traffic rates. If that probability is low enough, then the risk of deadlock will be low. Where concurrent transactions on a file are unlikely, it may be advantageous to either reject the transaction (let the user try again) or to call in manual assistance from, say, the Data Base Administrator to resolve the conflict. We believe that future work can be usefully directed to examining such pragmatic approaches.

Another area that appears to merit attention is the development of methods for identifying loss of (or impending threats to) data base integrity and for verifying that data base integrity has been (or has not been) compromised. For example, if a data base integrity threat alarm algorithm were developed, it could substitute for more costly preventive methodologies. The threat alarm function could monitor ongoing and offered transactions and either delay, prevent, or call for manual review of a transaction which threatens data base integrity.

Part of the distributed data base integrity problem is to periodically verify that data base integrity is preserved and to identify instances where integrity has been violated. This is an audit process which must run after the fact, using the transaction history. It would presumably be done during low

traffic periods and the transaction files, transactions or users to be audited would also presumably be selected on a random or on an activity-related basis.

Although data base integrity problems are related primarily to the concurrency of transactions (both updates and inquiries), the issues of outages in communications, processors, and data storage ("head crashes") must figure prominently in the various techniques of integrity assurance. Every activity must be planned on the assumption that any item of hardware can fail at any time before, during, or after any transaction. Indeed, there is a small but finite chance that multiple features, either independent or related, can occur. Any scheme designed to ensure data integrity must be able to operate in an environment of potential failure, or at least to be able to call for assistance in the event of failure. Presumably the more likely failure scenarios should be handled automatically by built-in programs and the less likely outages or combinations of outage would require operator intervention. An assessment of the chances of various types of failure and decisions on programmed versus manual handling should be made during the design process. The development of methods for carrying out this assessment, and the collection of useful data on failure probabilities in the real-world environment would be useful system design tools.

Hardware outages are not the only types of failure that the system will encounter. Failures can also be due to the characteristics of the data stream, and to programming faults which may occur only during certain data-related circumstances. As hardware becomes more reliable and software becomes more complex, it is increasingly important to focus attention on ways of assuring that software is both reliable and resilient when faced with a transaction stream. Experience with computer-based message switching systems (a distributed data base system has many of the characteristics of such a system) indicates that traffic handling capacity, service queues, and the essentially uncontrollable content of the transaction stream provides an environment that is prone to "software-type" failures. Other experience with airline reservation systems has shown that software failures can be as much as an order of magnitude more likely than hardware failure. As hardware continues to become more reliable, we believe it is essential that investigations be conducted in practical approaches to ensuring the reliability of software.

BIBLIOGRAPHY

1. Alsburg, Peter A. and Day, John D. A Principle for Resilient Sharing of Distributed Resources. Second International Conference on Software Engineering (ACM & IEEE Computer Society). October 1976.
2. Badal, D.Z. Data Base System Integrity. COMPCON Digest of Papers. Spring 1978.
3. Berg, John L. (Editor). Database Directions; The Next Steps. Proceedings of the Workshop of the NBS and ACM, Fort Lauderdale, Fla. October 1975.
4. Canning, Richard G. The Challenges of Distributed Systems. EDP Analyzer, vol. 16, no. 8. August 1978.
5. Champine, G.A. Current Trends in Data Base Systems. IEEE Computer, May 1979.
6. CODASYL Systems Committee. Selection and Acquisition of Data Base Management Systems. Association for Computing Machinery, March 1976.
7. Date, C.J. An Introduction to Database Systems. Addison-Wesley Publishing Co.
8. Davenport, R.A. Distributed Database Technology - A Survey. Computer Networks vol. 2, no. 3, July 1978. North Holland Publishing Co.
9. Deppe, Mark E. and Fry, James P. Distributed Data Bases; A Summary of Research. Computer Networks I (1976) p130-138. North Holland Publishing Co.
10. Everest, Gordon C. Concurrent Update Control and Database Integrity. Database Management (J.W. Klimbie & K.L. Koffeman, eds.). North Holland Publishing Co., 1974.
11. Fry, James P. and Maurer, John. Operational and Technological Issues in Distributed Data Bases. Data Communications Management, AUERBACH Publishers, Inc. 1978.
12. Goos, G. and Hartmanis, J. Lecture Notes in Computer Science. Proceedings of the 1st Conference of the European Cooperation in Informatics, Amsterdam, August 1976.
13. Hammer, Michael and Shipman, David. An Overview of Reliability Mechanisms for a Distributed Data Base System. Proceedings of COMPCON, Spring 1978.
14. Hardgrove, W.T. Distributed Database Technology: An Assessment. Information and Management 1 (1978), p157-167, North Holland Publishing Co.

15. Lefkovitz, David. Data Management For On-Line Systems. Hayden Publishing Co.
16. Maryanski, Fred J. and Fisher, Paul S. Rollback and Recovery in Distributed Data Base Management Systems. Proceedings of 1977 ACM Annual Conference.
17. Palmer, Ian R. Data Base Systems: A Practical Reference. Q.E.D. Information Sciences, 1975.
18. Peebles, Richard and Manning, Eric. System Architecture for Distributed Data Management. IEEE Computer, January 1978.
19. Ramamoorthy, C.V.; Ho, G.S.; and Wah, S.W. Distributed Computer Systems - A Design Methodology and its Application to the Design of Distributed Database Systems. Infotech International Ltd., 1979.
20. Ramamoorthy, C.V. and Krishnarao, T. The Design Issues in Distributed Computer Systems. In Distributed Systems (p375-400), Infotech State of the Art Report, 1976.
21. Rosenkrantz, Daniel J.; Stearns, Richard E.; and Lewis, Philip M. System Level Concurrency Control for Distributed Database Systems. ACM Transactions on Database Systems, June 1978.
22. Rothnie, James B. and Goodman, Nathan. A Survey of Research and Development in Distributed Database Management. Third International Conference on Very Large Databases. October 6-8, 1977, Tokyo.
23. Verhofstad, Joost S.M. Recovery Techniques for Database Systems. Computing Surveys, June 1978.

III. LOGGING AND RECOVERY

This section of the report discusses the state-of-the-art with respect to logging and recovery/rollback/restart logic within a distributed data base environment.

3.1 DEFINITION OF THE PROBLEM

The primary distributed data base capability to be provided by logging and recovery functions is the ability to restore a consistent and up-to-date data base* state after it has been destroyed by software, hardware, or human failures. Many secondary distributed data base capabilities (e.g., reports of data base access) can be provided with the data collected in the logging function.

Logging and recovery/rollback/restart capabilities are mandatory in any data base system environment. The ability to restore a data base after any failure that has destroyed all or a portion of that data base is a major data base system requirement. Providing this restoration service is even more complex within a distributed data base situation.

* Throughout this section, the term data base is used generically to include the actual data as well as associated directories, index tables, etc.

One of the main motivations for distributed data base systems is a requirement for high data base availability, that is, a need to ensure that a data base is nearly always accessible. Distributed data base systems seem to offer this characteristic since availability is not limited by the reliability of any single component but rather by the reliability of combinations of processing nodes and communication links in the network. These combinations can be configured to achieve arbitrarily high availability. In order to achieve this reliability, however, it is necessary that the distributed system be able to cope with the failures of individual components and continue operation.

To do this, the distributed data base system must be provided with a logging mechanism for use in recovery of lost files, in rollback, and to permit operation while certain nodes are unavailable. Interprocessor communications overhead can result in more time-consuming recovery and rollback operations if several processors are required to participate. It is also likely that within a distributed data base system there is a larger number of programs interacting than in a single machine system. Hence, the complexity and effect of any recovery or rollback procedure may be compounded.

When an executing procedure, an information processor node, or a communications link fails, data base recovery is often necessary. Where appropriate, data base updates that were only partially complete at the time of the failure must be reversed via rollback techniques, and the failed procedure(s) restarted to re-execute. If a procedure fails with updates or locks outstanding at multiple locations, cooperating recovery actions must be initiated. The condition becomes still more complex in the case of a processor node communications link failure in a distributed environment. A processor may fail while some of its resident procedures are indirectly updating remote data base elements and/or while remote procedures are updating local data base elements. Some action must be taken to synchronize recovery actions at all affected processors.

Most proposals agree that a distributed data base processing system should possess sophisticated software which facilitates recovery when either a user transaction crashes before completion, or a computer which is responsible for managing a portion of the data base crashes. Logging and recovery logic

may have to include consideration of the authorization of operations when one computer or a set of computers is missing because the computers are down or because the network has been partitioned into separate computer sets by communication failures. Provision of adequate techniques for reinserting these computers into the network when they resume operations must be considered. In any case, rollback logic requirements should be minimized.

3.1.1 Logging and Recovery Functions

Traditionally, the design of logging and recovery functions is one of the last data base management system capabilities to receive attention. Considering that distributed data base system design itself is in its infancy, one should not be surprised to find that there are many unanswered questions regarding overall logging and recovery/rollback/restart capabilities within a distributed data base environment.

Logging and recovery functions within a distributed data base environment require support tools (tools that are also used in centralized processing environments), such as:

- (1) Log (Journal/Audit Trail) - a secure secondary storage file which may contain such information as:
 - Before/After Control - for each change made to the data base or a distributed portion of the data base, a mechanism to record the physical record or updated part of it as it appeared just prior to and/or just after the change was made.
 - Transaction Log - maintenance of a copy of each transaction (event) or selected transactions that caused some discrete action.
 - Rollback System Commands - for each change made to the data base or a distributed portion of the data base, a system command that would reverse the change made to the data base.
- (2) Data Base Dump - periodic copying of parts of or the entire data base onto a secondary storage device which is independent of the production-mode device.

- (3) Checkpoint - a snapshot taken at prespecified intervals of the entire internal status of each of the distributed data base system's primary memories and associated components for recovery purposes.
- (4) Executable Code -
- Integrated Data Base Services - on-line modules integrated into the operating software that are designed to execute automatic recovery and restart of the entire data base or distributed portions of the data base.
 - Free-standing Data Base Utilities - self-contained utilities designed to be executed off-line in certain recovery situations.

3.1.2 Logging and Recovery Operations

In general, a permanent logging record should be made of all transactions which modify the data base in a distributed data base system. Optionally, other transactions (i.e., read only) might be logged. A chronological record of all transactions (or, optionally, only transactions which modify the data base) would be referred to as a log, journal, or audit trail. In addition to the actual transaction, a before and after data base control mechanism should be available to support automated recovery and restart. As an option, a suggestion has been made to log actual system commands that could be used via the distributed data base system to rollback, under certain circumstances, each actual update.

At certain points in time, data base dumps of parts such as a node's distributed data base or the entire data base must be stored on magnetic tape or other archival storage. Special checks may be carried out to ensure that the copies are error-free. This process could take a long time, possibly hours, if a large volume of data is involved. Transaction processing could not be active while the data base dump is being performed. Once a data base dump would be complete, a new version of the log must be started. A question here is whether the old log should be made part of the data base dump, and if so, with which version of the data base--the one just copied or the prior version.

Several unanswered situations include synchronization of the dumps (i.e., should all nodal data bases be dumped at the same time, and, if so, what can one do about inoperable nodes), and, if data is replicated, should the duplicated copies be dumped.

At other checkpoints, a copy of parts of or the entire primary memory and control tables of a node should be made, preferably to archival-type storage, for restart operations. This checkpointing process is also costly, and no user application may run during the checkpoint operation. Synchronization of checkpoints across nodes may also be necessary.

When part or all of a distributed data base is to be recovered, the following type of steps could be taken:

- (1) A determination would be made as to which node is or which nodes are to be restarted. The applicable node or nodes would not be available for transactions from any node or nodes still running.
- (2) All transactions destined for a node to be recovered would have to be stored elsewhere for future use.
- (3) The system fault, if any, causing the node failure would be repaired, if necessary.
- (4) The copies of the data base and memory made at the most recent dump or checkpoint would be loaded onto the physical devices on which the data base is stored. (This process of loading the data base may also require a substantial amount of time and memory.) After the loading is complete, the entire data or file system would be exactly as it was immediately after the last dump or checkpoint.
- (5) If before/after controls were saved, all appropriate before/after transaction controls on the log would be processed in order; the system will have recovered when this reprocessing is complete. Another mechanism would be to reprocess all appropriate transactions on the log file (this would be quite time consuming).
- (6) Incoming transactions which were stored during recovery would be processed.

Replicated portions of a nodal data base could be recovered by copying them from another node. Naturally, this situation would require interface logic with the log file restore operation.

Rollback of certain changes to the distributed data base could be performed by processing appropriate before/after control information in a reverse order against the then current data base, or by using reverse system commands described previously (if created and kept).

It should be noted that more coordination between the checkpoint information and logging information may be required. For example, the logging information processed prior to the time of the checkpoint may have to be applied prior to restoring the memory state to the checkpoint status.

A significant amount of time is required for data dump, checkpoint, recovery processes, and for maintaining the log journal. There are also costs for storage of the data base, checkpoints, and log data.

The recovery time increases with the number of logged items to be processed. Hence, the recovery time increases with the time between the most recent data base dump or checkpoint and the detection of an error. The greater the intercheckpoint or dump time, the larger will be the average time between a dump or checkpoint and detection of an error; thus, average recovery time increases with interdump or checkpoint time. If the time slice is too small, too much time is spent in dump or checkpoint processing, and if the time slice is too large, too much time would be spent during recovery. In a distributed data base, the problem of logging and recovery is more difficult than in a centralized data base as a number of concurrent operations may have to be performed at different processing elements.

Logging and recovery functions (either provided by one or more data base management systems--i.e., different systems may be used at one or more network nodes--or by other means) within a distributed data base environment require system and management support far in excess of those within a traditional centralized data base environment. Certainly, interactive users operating in a distributed processing environment pose some unique data base currency problems.

As a node may possess a data processing facility which contains data from other unrelated data bases, and that may be processing applications unrelated to the distributed data base environment, steps must be taken to isolate those data and processes from the distributed data base logging and recovery/rollback/restart logic.

3.2 DISCUSSION OF SOLUTIONS

The following paragraphs discuss possible approaches to:

- Logging
- Rollback and restart
- Other forms of recovery and restart.

3.2.1 Logging

Each individual node that contains and/or controls data base access appears to be the appropriate location to maintain the logs of such data base activity (although a centralized log concept has been proposed). If the concept of subtransactions is utilized (i.e., the node initiating the update request generates a transaction for each affected node), each subtransaction should be logged at each affected node; the originating source transaction may also be logged at the initiating node (again, here, varying viewpoints can be found). As a node processor may serve a large number of application tasks, the amount of recovery information should be minimized. Many existing single machine data base systems save the before and after images of each data base physical storage unit (e.g., page) that is altered. In order to reduce the amount of log file space required for such before/after control images, other techniques could be used. One suggestion has been to log actual commands of the distributed data base system that would rollback the update performed (unfortunately, this suggestion relies heavily on the integrity of the actual system which may itself have caused the failure, and also implies storage and processing costs for the rollback system commands—commands that do not remove the need for before/after control information needed for recovery after data base loss).

It has been suggested that the log file also contain restart information on all application tasks (restart information identifies a stable point at which an application program can be restarted). By default, the system should then

write a restart entry whenever an application task is initiated. It might also be desirable to permit the programmer the ability to indicate a restart point in a task (note that the CODASYL specifications do not provide a facility for this operation).

In certain cases, particularly a strict retrieval environment, the logging of read operations may produce a large number of entries on the log file that will never be applied in any rollback situation. One proposal suggests that considerable rollback overhead could be saved if the Data Base Administrator (DBA) were provided the option of shutting off the logging facility for the entire system or at selected nodes for retrieval operations, on, optionally, selective tasks, or if such logging were performed to a separate file. This could be accomplished at task initiation time. Note that this concept could also be extended to bulk update operations.

Actual logging must be done to a secure secondary storage file, possibly on a blocked basis; the log file associated with a data base must be coordinated with checkpoints of the file (e.g., time-stamps). If logging is done on a blocked basis (i.e., a block of log entries are gathered in core or on disk), some vulnerability in recovery/rollback is created as some number of log entries might be lost in the event of a failure.

It is also interesting to reconsider the problem of when to log, that is, before or after completing the actual data base updates (e.g., the final write of the data base, or the transfer of the shadow block (see 3.2.3 below) pointers). In a centralized environment, this problem is simplified versus a distributed environment where recovery/rollback may be utilized more heavily (e.g., in resolving deadlocks).

The logging area will also be influenced by data integrity requirements (see Section II). A problem faced by data integrity in the case of a replicated data base in which discrepancies are found between copies is to determine which version is the most or can be made the most current and accurate. Appropriate logging tools (e.g., time-stamps) would have to be considered.

3.2.2 Rollback and Restart

The rollback of an application task by a node or by many nodes (if affected) is an important element in any recovery scheme for a distributed data base management system. In a highly integrated data base, the procedure may be quite complex. The ability should be present to execute rollback concurrently on distinct nodes (however, a node processor could rollback only one application at a time).

When the distributed data base system software on a host node processor determines that an application task terminated abnormally, rollback procedures must be initiated, the host node processor would have to notify all affected nodes for this task that rollback must occur and provide the task name, initiation time, and shared data information. Each node processor that receives the rollback message must then refuse to accept any operations accessing the area updated by the terminating task. It has been proposed that the following rollback procedure could then be carried out at each applicable node:

- (1) The log file would be read "backward" to locate the initiation point of the task.
- (2) The log file would then be read "forward" from that point and the following operations, depending upon the entry on the log file, would be performed:
 - If the entry is an update entry for the task that is being rolled back, an entry would be made in the update list to indicate the record or set occurrence whose contents has been modified (an entry in the update list would consist of an ordered pair of record or set type and occurrence identifiers). The rollbacked data base file would be restored from the log file.
 - If another update entry for the task being rolled back alters the contents of a record occurrence that was previously restored (this is indicated by the presence of the record occurrence on the update list), no action should be taken with respect to either the update list or rollback data base file (this will ensure that an updated record is restored only once to its earliest value in the rollback procedure).

- (3) If an entry for an application task other than the primary rollback task references a record or set occurrence for which an update list entry exists, an entry for that task may also have to be rolled back (since the task may be operating with incorrect data). If so, the task name and time of this entry would be saved in a secondary rollback list.
- (4) When the log file has been processed up to the time of termination, the restoration of the portions of the data base affected by the primary rollback task would be complete.
- (5) The log file then is read "backward" in order to locate, for each task in the secondary rollback list, the restart point immediately preceding the time at which the incorrect operation was detected. Secondary rollback can then proceed.
- (6) Messages must be transmitted to the other node processors for the tasks in the secondary rollback list, indicating that the tasks should be rolled back to the specified restart point.

The rollback situation is obviously quite complex due to secondary processing logic. Other suggestions for handling this area, e.g., shared data lists which could be computed from the subschemas of application tasks, have been made.

3.2.3 Recovery and Restart

It has been proposed that in a distributed data base system a recovery procedure must be initiated, at various times, by:

- The computer node which crashed,
- The overall data integrity logic, or
- The overall distributed data base system logic if a communications failure is encountered and effects on transactions are suspected.

Local restoration of part of a distributed data base may imply global restoration; it is not certain that a consistent state can be reached otherwise. Solutions based on synchronized checkpoints, etc., have been proposed.

The idea of synchronized data base back-up is obviously of interest—however, it is not clear how to synchronize data base dumps, before/after controls, and checkpoints (e.g., if a node is busy or down). A possible

solution is to consider a method made up of a data base dump with before and after images to either be added or deleted from the data base dump.

One important design aspect that recovery can impact is deadlock handling. Due to potential inefficiencies in recovery of distributed data bases, it has been argued that deadlock prevention is more efficient than deadlock detection for a distributed data base system. However, an efficient recovery mechanism can make deadlock detection more attractive.

A proposed key to a recovery scheme appears to be the distributed data base system design notion of a shadow block. When a transaction updates a distributed data base block, the old block—called the shadow—should be kept for the contingency that a failure might occur before the transaction is completed (sort of automatic rollback). The shadow would be released only after the transaction had been properly logged and the new updated block had been properly constructed. After a new block has been constructed by an updating transaction, two versions of a block would reside on the back-up store, each representing part of a state of integrity of the data base.

To perform the update-commitment, the transaction must assure that the shadow would no longer be used as a shadow by other transactions. (This could be done by performing updates under a lock on the block.) Since at any time there might have two versions of a block, the distributed data base system would need two address maps defining for each block the physical locations of the shadow and the new block. These maps themselves would have to reside on the back-up store, and should be time-stamped, e.g., with the system-time of the moment of creation of the entry for recovery/rollback/restart purposes. The main difficulty of such a scheme in a distributed data base environment is the design of the update-commitment operation which performs the switchover from the shadow to the new block. (This operation itself may fail, and recovery must still be possible.)

3.2.3.1 Recovery and Restart as an Extension of Rollback

A suggestion that recovery could be approached as an extension of rollback (if the appropriate data base(s) is/are destroyed) has been made. There are three basic possible approaches in a distributed data base system:

- (1) Design the data base to permit only controlled or restricted interaction among application tasks.
- (2) Extend the single machine recovery mechanism to the distributed system. This approach would entail rolling-back all data bases on all backend node processors in the system to the point of initiation of the faulty task.
- (3) Use a selective recovery mechanism to roll-back only those tasks which have used data provided by the erroneous or terminated task.

The first approach, avoiding integration in the data base, has little appeal to the designer of a distributed data base system, although in practice this might be the most commonly used technique since many users of data base systems on single machines feel that avoiding integration is the only reliable means of ensuring data integrity. If this user philosophy were applied to a distributed data base system, very inefficient utilization of the distributed system would result. In order to avoid the need for a sophisticated recovery mechanism in a distributed data base system, two tasks would not be permitted to have simultaneous update capabilities to the same data base. For an application system under this requirement, the distributed data base system recovery problem could be simplified.

If the single machine approach were to be extended to a distributed data base system, then the entire data base would be unavailable during recovery. In a system with a large number of backend or bi-functional node processors, this approach could result in the recovery of a small portion of a data base while preventing usage of a large, correct segment of the data. The communication process involved with this technique would be that a message must be transmitted to all backend node processors indicating that recovery must begin. The time of initiation of the faulty program must be provided. This system-wide recovery approach would ensure that all effects of the erroneous application program would have been removed from the data base. The main drawbacks to this method would be that access to unaffected portions of the data base would be prevented and that unnecessary rollbacks might occur.

A selective recovery mechanism would overcome the main deficiency of the system-wide recovery strategy by rolling-back only those application tasks that would be operating with tainted data. Overall system throughput would increase under these circumstances, as would accessibility to the data base.

The communication overhead which potentially has the most significant performance effect in a distributed data base system ideally should not exceed one transmission to each backend node processor if the method is to be effective. The computational overhead involved in a selective recovery strategy should be maintained at a level where it is not significant in terms of system performance.

3.2.3.2 Recovery and Restart as a Stand-Alone Capability

In distributed data base systems which are highly integrated and support multi-threaded updating, a selective recovery technique would appear to be required if both performance and integrity are to be preserved.

The responsibility for recovery could be left to each node; local data base and directory updates could also be logged. In case of a file failure, the start of day file could be reloaded and the day's activities restored by processing the log tape. The failed node would be unavailable until the restoration is complete. A mechanism would have to be developed to inform the other nodes of the failed node's condition, so that processes do not hold up awaiting responses. This mechanism could be accomplished by a time-out function association with each request, or by an explicit message sent from a monitoring node indicating the unavailability of the failed node. Naturally, recovery could also proceed from any data base dump/checkpoint/log combination.

Those files that are crucial to system operation could be duplicated to assure continuous availability. This could be accomplished by placing redundant physical files and processors at the critical node; another approach would be to place copies of critical files at more than one node (this, of course, raises the problem of keeping these files consistent); a third approach would be to log the file updates to a neighboring node (if the original node fails, the file can be reconstructed at the neighboring node; the other nodes in the network would then be informed of the change in location of the critical file).

As previously mentioned, the proposed selective recovery methodology for distributed data base systems requires processing at data definition time, as well as run time. When a new subschema is requested by the Data Base Administrator (DBA) and created, a potential shared data list could be computed by intersecting all subschemas of that schema. The potential shared data list indicates record and set types that are in common with other subschemas.

Since each application task invokes exactly one subschema during its execution, the potential data overlap of any two application tasks could then be determined from their potential shared data lists. In order to maintain the data overlap information at execution time, the activation of a data base application task might result in a message being transmitted to all applications that have the potential to share data with that task. The message would indicate the application task and subschema names. The information relating active application tasks and subschemas could be maintained in the data dictionary. When an application task terminates, a similar message would be sent to all tasks with intersecting potential shared data lists.

Situations may arise in integrated distributed data bases in which application tasks share record types, but do not operate upon the contents of all data items in the record. A similar possibility is that the application task may access some data items in a read-and-print mode. In either of these situations, an incorrect value in a data item may not be critical to the function of the program. Rolling-back a task due to an incorrect value in a noncritical data item would have an adverse effect on system performance.

The identity of noncritical data items is heavily application task dependent and can in no way be inferred from a subschema description. Since the CODASYL specifications do not provide for the identifications of noncritical data items for recovery purposes, some additional mechanism for their identifications should be provided to the Data Base Administrator. The simplest approach would be to maintain in the data dictionary a list of noncritical data items for each application task.

When subschemas are intersected to form the potential shared record list, noncritical data items should be removed from the intersection of the records. Only those records which intersect on critical data items should be included in the potential shared data list.

3.3 ASSESSMENT OF THE STATE-OF-THE-ART

Advances in distributed data base systems are among the most significant developments in current computer technology; however, techniques for the analysis and design of distributed data base systems are in their infancy. Although some consideration has been given to update philosophy (including deadlock and data integrity), less work has been done on other areas (such as security, privacy, logging, and recovery/rollback/restart).

The basic issues today in the area of distributed data base systems are similar to those that have faced traditional data base systems for the many years. Issues concerning centralized versus decentralized data, level of redundancy (multiple/replicated copies), privacy, integrity, and security existed long before the advent of distributed data base technology. These issues are, however, further complicated by the autonomous and independent nature of a distributed data base system. For example, these issues increase considerably in complexity when problems involving multiple-copy data files and nonfunctioning host computers are introduced.

Specifically, in regard to logging and recovery/rollback/restart techniques within a distributed data base environment, little, if any, implementation effort has been expended, and limited design time has been provided. Coupling the fact that logging and recovery/rollback/restart techniques have usually been retrofitted into data base systems, with the fact that overall limited advances in distributed data base systems design have been completed, indicates that additional research and development (R&D) effort must be spent in the logging and recovery/rollback/restart areas within a distributed data base environment.

3.4 AREAS FOR FURTHER STUDY

In general, several areas require additional research prior to optimum use of distributed data base systems. Further integration of nodes within resource-sharing systems is required in order to provide a foundation for distributed data base systems. This area would involve the transferability of data, transparency of processes from dissimilar nodes, and the distribution of resources (data and software) to optimize system performance. Synchronization of multiple copies of distributed data must also be investigated. Problems with update, back-up, and concurrent access increase in complexity due to the distributed environment, and must be fully analyzed. Finally, the capability of the resource-sharing and distributing systems to store data and execute programs at any node will depend on the development of data base management system translation technologies--particularly data query and model translation. These technologies must be developed in order to achieve the integration of data base management systems, which is an initial goal of distributed data base systems.

The following areas are specifically recommended for further study:

(1) Distribution:

- Data:
 - Complete copy at each node
 - Unique copy at each node
 - o No replication
 - o Replication
- Directories:
 - Centralized at one node
 - Unique copy at each node
 - o Centralized control
 - o No centralized control.

(2) Logging:

- Where to log:
 - At node receiving transaction
 - At each node handling transaction
 - At each node where transaction causes handling of data

- What to log:
 - Transaction
 - o All
 - o Update only
 - o Selected
 - Before/after image for update
 - Reverse data base commands
- Physical:
 - What medium
 - Blocked vs. unblocked
- Node failure:
 - Where to keep transactions
- Application tasks:
 - Restart points
 - Subschema data utilization
 - o Across nodes
 - o Across applications
- When to log:
 - Before data base access
 - After data base access
- Time-stamps
- Data base locking considerations.

(3) Rollback:

- Prevention
- Across nodes
- Secondary rollback
- Shared data lists from subschema data base access.

(4) Recovery:

- Extension of rollback philosophy
- Selective by:

- Application
- Node
- System

- From:

- Backup
- Other nodes/files
- Logs.

(5) Role of Data Base Administrator

(6) Synchronization/Consistency/Integrity of Updates:

- Data and directories:

- Loss of message
- Replication of message
- Process disappearance

- Control:

- Centralized/decentralized
- All or no transaction applied
(if node down, security/privacy problems, etc.)

- Reliability of data and directories.

(7) Restart:

- System
- Node
- Application.

(8) Failure Detection:

- System
- Node
- Data base
- Communication
- Deadlock.

BIBLIOGRAPHY

1. Adiba, M.; Chupin, J.C.; Demolombe, R.; Gardarin, G.; and LeBihan, J. Issues in Distributed Data Base Management Systems - A Technical Overview. Proceedings of the Fourth International Conference on Very Large Data Bases, p89-110, Grenoble 1978.
2. Alsberg, P.A. and Day, J.D. A Principle for Resilient Sharing of Distributed Resources. Proc. Second International Conference on Software Engineering, p562-570, 1976.
3. Aschim, F. Data Base Networks - An Overview. Management Information vol. 3, no. 1, February 1974.
4. AUERBACH Publishers, Inc. Distributed Data Base for Distributed Processing. AUERBACH Computer Technology Reports, 1976.
5. AUERBACH Publishers, Inc. Functions of Data Base Administration in Production Mode - Part II. AUERBACH Data Base Management Reports, 1976.
6. AUERBACH Publishers, Inc. Trends in Data Base Technology, AUERBACH Data Base Management Reports, 1977.
7. Bayer, R. Integrity, Concurrency and Recovery in Data Bases. Proc. of the International Conference of the European Cooperation in Informatics. August 1976.
8. Booth, G.M. Distributed Databases - Their Structure and Use. Distributed Systems Infotech State of the Art Report, 1976.
9. Chandy, K.M. et al. Survey of Analytic Models for Rollback and Recovery Strategies in Data Base Systems. IEEE Transactions on Software Engineering, March 1975.
10. Davenport, R.A. Distributed Database Technology - A Survey. Computer Networks vol. 2, no. 3, July 1974. North Holland Publishing Co.
11. Gelenbe, E. A Model of Rollback/Recovery with Multiple Checkpoints. Proceedings of Second International Conference on Software Engineering, October 1976.
12. Hebalkar, P.G. and Jung, C. Logical Design Considerations for Distributed Data Base Systems. Proc. COMPSAC '77, November 1977.
13. Lampson, B. and Sturgis, H. Crash Recovery in a Distributed Data Storage System. Technical Report, XEROX Palo Alto Research Center.
14. Liebowitz, B.H. and Carson, J.M. Distributed Processing, IEEE Computer Society, 1978.

15. Maryanski, Fred J. and Fisher, Paul S. Rollback and Recovery in Distributed Data Base Management Systems. Proceedings of 1977 ACM Annual Conference.
16. Ramamoorthy, C.V. and Krishnarao, T. The Design Issues in Distributed Computer Systems. In Distributed Systems (p375-400), Infotech State of the Art Report, 1976.
17. Ramamoorthy, C.V.; Ho, G.S.; Krishnarao, T.; and Wah, B.W. Architectural Issues in Distributed Data Base Systems. ACM, IEEE Proceedings of Conference on Very Large Data Bases (p121-126) Tokyo 1977.
18. Rothnie, J.B. and Goodman, N. A Survey of Research and Development in Distributed Database Management. Third International Conference on Very Large Databases (p48-62) 1977.
19. Thomas, R.H. A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases. ACM Transactions on Database Systems, p180-209, June 1979.
20. Thomas, R.H. A Solution to the Concurrency Control Problem for Multiple Copy Databases. Digest of Papers, COMPCON '78 p56-62, March 1978.
21. Thomas, R.H. A Solution to the Update Problem for Multiple Copy Data Bases Which Uses Distributed Control. Bolt, Beranek and Newman Tech. Rep BBN-3340, July 1976.

IV. DEADLOCK

This section discusses distributed data base system deadlock resulting from concurrent, conflicting needs for data base resources.

4.1 INTRODUCTION

An objective of computer system designers is to improve the utilization of system resources. One approach is to distribute resources among concurrently executing tasks.

Requests by separate tasks for resources may be granted in such a sequence that a group of tasks is unable to proceed, because each task is monopolizing resources and is waiting for resources currently held by others in the group. The tasks are then deadlocked.

The resources may be files or other designated portions of the system data base. The number of tasks in the deadlocked group may be two or more. Deadlock is a logical problem and it can arise in many contexts.

When deadlock occurs, the progress of the involved tasks is halted. Until some action is taken, the deadlock will persist. Thus, it is necessary to prevent deadlocks from occurring or to resolve deadlocks that do arise. This necessity prevails despite the rarity of deadlocks.

In distributed data base systems, as in other computer system configurations, the same logical problem exists. Dealing with it, however, is much more difficult in this context.

4.2 THE DEADLOCK PROBLEM

Deadlock can occur in any computer system if concurrent users are forced to wait for each other. Concurrent use alone does not cause deadlock; it is a side effect of concurrency control.

4.2.1 Concurrency Control

Any data system allowing multiple users of a data base to be active concurrently must protect those users from each other. In general, a task should produce the same results in a concurrent environment as it would running alone. In particular, its data base operations—reading and writing—should not be affected by the presence of other tasks.

A simple data processing system must provide for concurrent use of the data base by all of its tasks. A distributed data processing system must consider concurrent operation of the tasks entered from and processed at each node, with respect to the system data base. In a distributed data base system, consideration must also be given to the data bases at each node, including replicated portions which exist at multiple nodes.

The concurrency issue with respect to a data base arises because of update. It is obvious that reading during update is hazardous: the reading task may read data that is partially updated. Concurrent update must also be prevented. Consider, for example:

- Tasks A and B both intend to update records 1 and 2 in a file

- Task A updates 1; next, B updates 1 and 2; finally, A updates 2
- The data base contains A's version of 2 and B's version of 1, and is now inconsistent.

This scenario could easily occur in a distributed data base system; if the tasks and records are at different nodes, the sequence of updates may be unpredictable. Furthermore, if files are replicated, the update sequences could vary among copies of files. Therefore, some form of concurrency control is required to prevent errors due to update.

4.2.2 Locking

The most common concurrency control technique is locking (although there are also other techniques). Locking effectively gives exclusive control of a resource to one user, and other users are thereby prevented from accessing the resource until the lock is removed.

Before a task is permitted to begin updating, a lock may be applied to the data base or a portion thereof. When the task completes its work, the data portion would be unlocked. In the interim, no other task would be able to acquire and lock the same portion. Similarly, a reading task that requires protection from update interference could lock the data portion that it intends to read.

Any task encountering a lock while attempting to access the data base would be unable to proceed, pending unlock.

Operating systems frequently offer locking capability at the file level. Finer (or coarser) locking granularity is also possible, including:

- The entire data base
- Records
- Data items in certain files
- Data items in certain records
- Other sets of data items or instances of such sets.

Locking granularity affects control complexity. Control is also more complex in distributed data bases. For example, locking/unlocking of data portions may involve:

- Task and data at different nodes
- Data portion stored in part at several nodes
- Data replicated at multiple nodes
- Combinations of the above.

4.2.3 Occurrence of Deadlock

It is conceivable that a task could require exclusive control (locking) of more than one portion of the data base (say, files). The task (Task 1) could lock File A and request File B but find B already locked out by another task (Task 2). At the same time, Task 2 could be awaiting release of File A. The two tasks would wait for each other—forever. This is deadlock. (Deadlock has also been called deadly embrace.)

Deadlock results when resources are assigned to processes, and each process cannot continue unless it is assigned a resource which another process is using. It can involve any number of processes (tasks) >1 and any number of resources (data base portions) >1 .

Deadlock, when it occurs, is a serious problem. The impasse will not resolve itself and the tasks will remain uncompleted indefinitely. The locked data portions will also remain unavailable indefinitely.

4.2.4 Deadlock in Distributed Data Base Systems

Deadlock in a single system is well understood. In distributed data processing, the locking problem is made more difficult by the existence of multiple centers of control. If no one processor controls all others, concurrency must be controlled through cooperating algorithms at each node. In a distributed data base, deadlock can result from two tasks in different processors accessing data at two other nodes, and more complex situations can be imagined.

The system control must prevent deadlock or must recognize that it has occurred and resolve it. Deadlock must be distinguished from other delays, such as a task that runs forever. Resolution of deadlock must not, after a deadlock-free period, result in the same deadlock and an interminable cycle. Deadlock prevention must not prevent a task from ever being run.

Deadlock is not necessarily a frequent occurrence. One installation reported two deadlocks in a year of multiuser operation; this was considered not enough of a problem to warrant any attention. However, it is felt that as distributed data base systems proliferate and the numbers of concurrent users grow, deadlock will become a more important consideration.

4.2.5 Approaches to Solutions

In any case, it must be recognized that deadlocks can occur and the problems must be dealt with. Deadlock can be prevented in conventional systems at some cost. (It is said that deadlock is prevented in associative processors by giving a task exclusive control of the processor's data base; the validity of this concept to distributed data bases is not clear.) Detection of the existence of deadlock is also possible, and there are many detection algorithms.

Detected deadlock must be resolved. One of the deadlocked tasks must be aborted (even if it is done manually) and rerun later. The partially completed task must be backed out of the system using techniques resembling recovery from system failure. It is suggested that the resolution of deadlock in a distributed data base is so difficult that deadlock prevention is better than deadlock detection. However, it may be true that if recovery procedures are adequate, deadlock resolution mechanisms are already available.

There is an extensive body of literature on deadlock in data systems, much of it offering solutions to the less difficult aspects of deadlock. Very little has been done in extending the research into distributed data base systems, other than to point out the additional complexities. Some papers deal directly with the problem, but the discussions do not report the results of implementations. Thus, the concepts in this paper are basically untested in a distributed data base environment.

4.3 SOLUTIONS TO THE DEADLOCK PROBLEM

The literature offers three basic approaches to dealing with the problem of deadlock. In general, these approaches may be classified as:

- Prevention
- Avoidance
- Detection and resolution.

These approaches were initially proposed for configurations that are simpler than those found in distributed data base systems.

Prevention calls for designing a system so that there is no possibility that deadlock will occur. This would be an attractive approach if it did not sacrifice important system features. One simple method of prevention is to permit only one job to use the system at a time, with job completion a prerequisite to execution of another job. Obviously, with only one job extant, deadlock is by definition impossible. This approach sacrifices concurrent use of the system.

A more practical solution prevents deadlock by assigning all the resources needed by a job (task) prior to its initiation. The resources become effectively locked out, since they may not be assigned to any other job until released. Deadlock is thus prevented but there may be execution delays--waiting for resources to be assigned--even though deadlock might not have been logically possible. The extent of execution delays depends upon the degree of granularity in data base assignments to tasks.

Avoidance of deadlock involves job executions except when deadlock is possible. The possibility of deadlock must be continuously evaluated. All uncompleted transactions must be capable of completion without deadlock; any resource assignment that would permit deadlock must be prohibited and the requesting task suspended until a safe state occurs. Substantial inter-node communication and processing overhead may be incurred.

Detection recognizes that a deadlock exists and calls for resolution of the situation. Detection is not difficult if the status of each node is known, but resolution requires eliminating at least one competing task. Since a task may not be eliminated permanently, it must later be restarted or rerun. Rerun first requires backing out the task (rollback) at all involved nodes, so that redundant processing will be prevented. Restart would be even more complex.

A manual approach to detection calls for the operator to observe the occurrence of deadlock and to take action as necessary to resolve the problem. In a distributed data base environment, this approach may not be feasible.

4.3.1 Deadlock Prevention

Deadlock can be prevented by designing a deadlock-free system. In such a system, the ingredients necessary for deadlock to occur would not all exist at the same time, and deadlock would then be impossible. The five necessary ingredients are:

- Lockout -- exclusive control
- Concurrency -- competing tasks
- Additional request -- for exclusive control
- No pre-emption -- no forced release
- Circular wait.

A system design that does not permit any one (or more) of the five conditions is sufficient to prevent deadlock from ever occurring. Solutions to each of these are now considered.

4.3.1.1 Lockout

Lockout is the most widely used solution to the problem of preserving data integrity during update. Despite the complexities and performance degradations imposed by locking out portions of the data base, this is the approach usually proposed for a distributed processing system. Therefore, if concurrent update is to be permitted, lockout is probably required.

Conceivably, update could be prohibited in some environments, so as to eliminate the need for lockout. Elimination of update in this context is a trivial suggestion; this paper addresses update problems.

Lockout because of update is a concern only with respect to that which is updated. For example, consider two update tasks, one at each of two nodes, and each task wishes to update¹ its local data base. Assume no replication of data. It is clear that the update processes do not conflict and cannot, of themselves, lead to deadlock. (It is assumed that no data dependent or secondary updates by the same tasks are triggered during execution.) Concurrent update can cause deadlock only if the same portion of the data base is updated.

¹In this discussion we speak only of update, although if one process, for example, reads what another process updates, the same need for lockout prevails. Other combinations of tasks are also relevant.

In this context, a portion of the data base refers to the level at which lockout is applied. If files are locked, deadlock can result from concurrent update of a file; if locking is at the data element level, deadlock could not occur unless the concurrent processes were to update the same data element. Thus, it is demonstrated that, while lockout is a necessary ingredient for deadlock, the locking granularity affects the significance of lockout.

Granularity at the coarsest level implies lockout of the entire data base whenever an update task is initiated. No concurrent task using the data base would be permitted. Deadlock would be prevented at minimal cost in complexity or overhead. In a batch processing environment, this approach might be acceptable, but it is obviously not suitable for a distributed processing network.

Finer granularity offers increased opportunity for concurrent processing without the possibility of deadlock. However, as granularity becomes finer, complexity increases.

- (1) Lockout complexity increases dramatically with progression from files to records to data elements, or to subsets at those levels.
- (2) Specification of the granule to be updated is more difficult when fine. If the applications programmer is responsible for granule specification, ordinary JCL or, at most, an OPEN statement should suffice for locking at the file level, while finer specification places a new burden on the programmer.
- (3) File level locking might be managed via a table that is replicated in each processor. With finer granularity, such a table might be too large to replicate, and locking control would have to reside at the nodes of the locked granules. Inter-node communications would probably be more complex.

4.3.1.2 Concurrency

Concurrency, the second of the five necessary ingredients, refers to two or more active processes or tasks which could compete for two or more (lockable) resources. (As before, "resources" means portions of the data base.) The deadlock problem statement in paragraph 4.2 shows that deadlock is possible only with at least two tasks and two resources. It is also clear that the resource requests must be distributed among the tasks; a task that does not require resources cannot cause a deadlock.

Competition can be destroyed by serial treatment of tasks for purposes of resource allocation. This is probably the most straightforward approach to deadlock prevention. It involves:

- Assignment of all needed resources before beginning execution
- Postponement of execution if all resources cannot be assigned
- Release of all assigned resources upon postponement.

This approach is inherently weak in any processing environment because it:

- Requires knowledge before execution of all resource requirements; e.g., it cannot accommodate a need for data that is undefinable prior to a data base access
- Holds resources longer than necessary---at least from request time until the time the resource is actually used.

In using this approach, the weaknesses must be tolerated. All data requirements must be predefined, and tasks must be designed without data dependent requirements. There would have to be a means for identifying the data needs prior to execution; presumably, this would impose some workload upon the programmer. As pointed out above, locking granularity affects the programming difficulty involved.

The assignment of a task's resources must be essentially a continuous process. If it is interrupted by the assignment of resources to another task, the concurrency issue remains unsolved. In a single jobstream, it is simple to make these assignments. In a distributed data base system, assignment at various data locations could not be a continuous, uninterruptible process without imposition of substantial delays and sacrifice of system capacity. Therefore, a method is needed to achieve the effect of assignment of resources to one task at a time.

An algorithm proposed in [1] offers a two-stage reservation process:

- Messages requesting lockout are sent to the nodes controlling the needed data

- If all requests can be satisfied, the requests are confirmed and the task is permitted to proceed; if not, the requests are cancelled and the task is put in a wait state.
- Conflicts among requesting tasks are resolved by a task priority scheme.

It is claimed that this algorithm will always work, but will have a considerable amount of overhead.

In [5], a similar mechanism is suggested, but the requests are passed from node to node along a fixed path. This preassigned order of nodes eliminates the need to assign task priorities and is said to improve deadlock prevention efficiency. It is not clear if, at any node, the processing of a request for multiple resources may be interruptible by another task's request.

The concept of task priority--to resolve request conflicts--is not trivial, and many methods have been suggested. Any numbering method that ensures unique task numbers would suffice, but a method that allows numbering at the input node, with no need for inter-node communications, would be preferred. A user identification number might be used, although it is not clear that more than one identically numbered task could not co-exist. Node number concatenated with (local) clock time--or perhaps with a consecutive task number--would ensure uniqueness.

There is a strategy that avoids the need to prioritize tasks. Instead, files are assigned unique numbers, and requests are processed in order of file number. This simple scheme prevents deadlock. However, it may involve considerable communications, since the relationship between file numbers and nodes may lead to multiple messages to a node at different times. Its applicability where locking granularity is finer than the file level has not been explored.

It must be emphasized that this entire discussion of eliminating concurrency is based upon preassignment of all needed resources before execution. This approach is inherently inefficient since it ties up portions of the data base longer than is necessary. It also prevents the system from processing tasks that do not completely identify data needs in advance.

A more efficient method has been proposed requiring preidentification of data needs but not preassignment. Execution is allowed to begin and availability is not assessed until the first data access is attempted. At this time, all files (or finer granules) preidentified by the task are assigned if available, or execution is suspended. Note that this allows execution to proceed and files to be used by other tasks for some period of time.

In this proposed method, tasks are grouped into sets of tasks which use some of the same files. The sets are continually redefined as tasks arrive or are completed. Deadlock could only occur within a set. File availability can be assessed by consideration of the tasks in the set, ignoring all others. Furthermore, control of a set is allocated to a designated node, further simplifying the control process. A task may release a file, even though it may need it again later. The set controller could then assign the file to a waiting task, permitting concurrent execution without undue complexity.

This mechanism is described in detail in [5]. It is admitted to be not easily implemented and to require considerable system overhead when continually forced to re-examine tasks to determine if requests can be allowed.

4.3.1.3 Additional Request

This is the third of the five necessary ingredients for deadlock to occur. It refers to a task issuing a request for a locked out data resource while the task already holds a locked data resource. It was shown in the paragraph 4.2 discussion that deadlock occurs only when such an additional request is issued. If all data requests are issued by a task at one time, deadlock cannot result.

Deadlock caused by additional requests can be eliminated by the assignment of all data resources before execution in the same manner discussed previously. In some systems, this solution may not be acceptable. An alternative approach to additional request may be appropriate for deadlock prevention.

The solution is simply not to permit additional requests. That is, any time a task requests resources it must first relinquish all resources presently held. Locked out data portions must be unlocked before new requests can be accepted.

This condition obviously applies if all requests are issued before execution. Once execution is initiated, if a new data need is identified, the task may or may not be able to release the portions presently locked, depending upon the characteristics of the task. If the operating system automatically releases locked portions when new requests are issued,

- The task may become suspended because another task seizes the same portion and the first task still needs access to it, and
- When the task resumes, the data may have been updated and completion of the task may produce incorrect results.

The burden appears to be placed on the programmer to ensure timely request/release of data resources. A feasible approach may be to divide tasks into steps, and to issue requests (e.g., via JCL) before each step, with automatic release at the end of each step. This approach is identical to the concurrency solution, except it operates on steps instead of tasks. Of course, it requires that a task be divisible into steps that can be executed without regard for data base changes between steps. Also, it requires, as in the concurrency solution, that all data needs be identifiable prior to execution of a step.

In the processing of a step's data requests, all the requests must be handled without interruption for processing some other step's (task's) requests. Once again, the concurrency discussion applies, and the asynchronous request processing at the various nodes must be handled, using the same techniques.

If step level assignment is feasible, it is a more efficient technique than task level assignment, since resources would not be needlessly tied up during steps that do not use them. On the other hand, the overhead associated with request processing for each step is greater, especially if the same resource is used by several steps. It appears, however, that complexity and overhead would be reduced if each step (or many of the steps) used only data at one node. This consideration could influence the data distribution philosophy employed in designing the network.

4.3.1.4 No Pre-Emption

The fourth necessary ingredient is no pre-emption. This means that deadlock can occur only in a system that does not permit a data request to force the release of data already locked out. If a task were able to pre-empt another's data assignment, it would not need to wait (interminably) for the resource to be released; deadlock would be avoidable.

The solution to no pre-emption is pre-emption. A task would have the ability to override another task's lockout. Presumably, the override would be conditional; e.g., only when deadlock might occur or has occurred. Presumably, a priority scheme would identify the pre-emptor and pre-emptee.

It is conceivable that pre-emption could be feasible in some situations, as when data has been assigned but not yet accessed. A flag of some sort could identify such a condition. Whether or not a pre-emption of this nature--after execution has begun--can affect the validity of results has not been addressed.

In the context of update--where data is undergoing modification--it appears that pre-emption cannot be permitted. Pre-emption in an update environment is not suggested in the literature.

4.3.1.5 Circular Wait

The fifth and last necessary ingredient, circular wait, is the distinguishing characteristic of deadlock. It is represented by a continuous loop of tasks in which each task holds at least one resource being requested by the next task in the loop. A minimal loop contains two tasks and two resources, but the numbers could be much larger. In this paper, the resources of interest are files or finer data granules.

Circular wait can be prevented by introducing a discontinuity into the loop, i.e., forcing the string of tasks to be linear, not circular. This can be accomplished by imposing a linear order upon the data base such that resource requests are always considered according to a predefined sequence. For example, assume each file has a unique sequence number. When a task requests (locked) assignment of several files (or portions within several files),

the files would be assigned serially, in the order prescribed by their respective sequence numbers. If a file is already locked for another task, the requesting task must wait, and no files farther down the list may be assigned before the locked file becomes available. This way, a task cannot have control of one file while awaiting the availability of a file with a lower sequence number. Thus, circular wait is not possible.

It should be noted that this procedure does not require all files to be assigned before execution of a task may begin. Nevertheless, if all requests are made at one time, they may be sequenced by the operating system without any concern on the part of the (task) program. If, instead, the program requests assignments at different times during execution, the programmer must ensure that the requests are issued in the sequence prescribed by the file numbers. This is a severe discipline to impose on the programmer. Furthermore, developing a preordering strategy (for files) that would be suitable for all or most programs is probably infeasible in a large data base.

4.3.2 Deadlock Avoidance

Another alternative to prevention of deadlock is avoidance of situations where deadlock might occur. In avoidance, the system prevents the two or more processes which could become deadlocked from running concurrently.

It is true that for any set of unexecuted tasks there is at least one sequence of task execution that cannot lead to deadlock. At worst, the tasks could be run serially. Any two or more tasks that do not use any files in common may be run concurrently without danger of deadlock. An execution sequence is safe if it avoids the possibility of deadlock.

Models have been proposed for the analysis of tasks and their resource requirements, leading to the identification of safe sequences. (Selection of the most desirable sequence from a set of safe sequences is a separate problem and is not addressed here, although some straightforward approaches can be conceived.) Presumably, the model should be executed whenever the mix of tasks or resource requirements changes, i.e., when a task is entered or completed or when a resource is released.

For a model to operate upon a complete list of tasks and files, centralized control is an obvious necessity. Furthermore, dispatching of tasks in accordance with a derived sequence also implies centralized control. The flow of control information throughout the network and the model computation frequency suggest an enormous amount of overhead in a distributed system.

This approach to avoidance requires that all data needs be identified before execution. Thus it is subject to the same weaknesses as are evident in prevention approaches with the same requirement. It is considered very unattractive in distributed data base systems because the necessary advance information is either not available or is distributed so widely as to cause considerable overhead and delay in any attempt to avoid deadlocks.

A slightly different approach has been suggested whereby files are assigned to tasks only when there is no risk of deadlock. No details are available, but it seems that a similar analysis of tasks and resource requirements would have to be performed to assess risk. It appears that this suggestion suffers the same weaknesses, without the benefit of generating safe task sequences. In theory, at least, it would also avoid deadlock.

Most of the literature does not discuss avoidance as a solution to deadlock. It appears to have little potential value to distributed data bases.

4.3.3 Deadlock Detection and Resolution

If a distributed data base system design allows deadlock to occur, it must be assumed that it will occur. Detection and resolution are the means of dealing with that eventuality. Each must be addressed, since:

- In a distributed data base environment it may not be at all obvious at any one node that a delay is due to deadlock, so some device is needed to test for and confirm its existence, and
- Detection alone only identifies the problem; action is necessary to allow the system to resume full operation and to complete the deadlocked tasks.

4.3.3.1 Detection

As an exercise in logical analysis, deadlock detection seems to have great appeal to computer science researchers. Numerous algorithms have been proposed for testing and proving that deadlock in fact exists. Most are based on maintenance of some form of state graph which is examined to determine if a circular delay exists. The algorithms' proofs are generally direct, and there is no doubt that correct techniques are available.

With centralized control, all the information needed to test for deadlock is available at the controlling processor--task status, resource requirements, resource status. In a distributed data base system without centralized control, no processor has enough information to detect all deadlocks, although a deadlock wholly contained at one node is probably identifiable. It is suggested that each node manager could broadcast local status to all other managers, and each would then assess the global picture. Both the centralized and this decentralized method involve substantial overhead and may not be acceptable if locking granularity is small.

Despite the overhead, detection offers an important advantage that is difficult to obtain with prevention methods: there is no need to specify (and assign) all data requirements in advance. An applications program may proceed to develop its data needs throughout task execution--based on input data, file data, computation, etc.--and to request and release data at any level of granularity without the imposition of deadlock prevention restrictions. This is a valuable characteristic.

It should also be recognized that detection overhead is not necessarily a costly feature. Deadlock is in many systems a relatively rare occurrence. Frequent exchange of status data and execution of a detection algorithm is not essential. In fact, it may be performed only when a task has been waiting unduly for a file; this seems to be a realistic approach.

4.3.3.2 Resolution

Resolution is much more complex than detection. When deadlock occurs, the responsible tasks are unable to continue execution, and some action is necessary to eliminate this condition. The common solution is to remove one of the contending tasks; the others may then proceed, free of the deadlock. A set of priority rules is needed to select the task that will be removed.

Deadlock resolution does not end with the elimination of the deadlock. Since a task has not been permitted to run to its conclusion, the abort must be dealt with. In the first place, the resources held by the aborted task must be released; this obvious step is necessary for the deadlock to be eliminated. Secondly, the task must be restarted at a later time. How to reschedule it without reinitiating the same deadlock again and again has been addressed by a numbering method proposed in [21]. The method prevents restarting forever, but it does not appear to prevent restarting an intolerable number of times.

The third and most complex aspect of task abortion is the need to back out the task; i.e., to undo what has been done. If a task is to be restarted, the restart should produce the same results that would have been attained had the task been initially able to end normally. In particular, the data base must be restored to its pretask condition.

Backing out of a task as a result of deadlock is similar to recovery from mid-task failure from some other cause. The effect of any data base changes must be nullified at every affected node. If the task generated other tasks--to update files at other nodes or to trigger data-dependent processes--these must be cancelled; if already initiated, they, too, must be backed-out. Recovery from an aborted task is discussed at length in Section III (Recovery).

Backout (rollback) because of deadlock is immensely complex in a distributed data base environment. However, it does not appear to be any different than backout because of failure. Consequently, if recovery is handled within a system design, the same facilities can be used for deadlock resolution.

4.4 ASSESSMENT OF THE STATE-OF-THE-ART

Deadlock is a real problem in multiuser computer systems. Distributed data base systems are more subject to deadlock than are simpler configurations, because they typically have:

- More concurrent users and
- Replicated portions of the data base.

Since locking is commonly imposed to preserve data base integrity during update, deadlocks are prone to result. Until a satisfactory alternative to locking is developed, deadlock will remain a problem. In distributed data base systems, prevention and resolution of deadlock can be enormously complex and costly.

Deadlock is an intriguing problem, and it has interested numerous researchers. Even in the context of distributed data base systems, a young field within computer science, deadlock has been given some attention. The problem is well understood, and the applicability of common solutions has been considered. In general, it has been concluded that the solutions would work, but the cost would be very high.

The literature is virtually devoid of implementation experience. The feasibility of solutions has not been tested on real distributed data base systems. The cost and complexity have not been measured.

Deadlock is presumed to be a relatively rare occurrence. No work has been reported in measuring or predicting the frequency of deadlock. Without data on the potential cost of deadlock, there is little guidance available to help allocate resources to the problem. For the same reason, there is no basis for describing solutions as costly.

4.5 AREAS FOR FURTHER STUDY

There is a good theoretical base for the study of deadlock. More work can be done on deadlock in distributed data base systems, but it can only keep pace with research into other aspects of distributed data bases. There is little need for more algorithms to detect the existence of deadlock.

Empirical data is badly needed. Studies should be undertaken to:

- Develop algorithms for predicting the frequency of deadlock
- Measure the impact of concepts such as finer locking granularity on the frequency of deadlock
- Estimate the cost of deadlock prevention methods
- Estimate the cost of deadlock detection and resolution
- Measure the frequency of deadlock in operational installations and develop empirical cause-and-effect relationships.

BIBLIOGRAPHY

1. Aschim, F. Data Base Networks - An Overview. Management Informatics vol. 3, no. 1, February 1974.
2. Bayer, R. Integrity, Concurrency and Recovery in Data Bases. Proceedings of the 1st Conference of the European Cooperation in Informatics, August 1976.
3. Chamberlin, D.C.; Boyce, R.F.; and Traiger, I.L. A Deadlock-free Scheme for Resource Locking in a Data Base Environment. Information Processing 74. Proc. IFIP Congress, August 1974.
4. Champine, G.A. Current Trends in Data Base Systems. IEEE Computer, May 1979.
5. Chu, W.W. and Ohlmacher, G. Avoiding Deadlock in Distributed Data Bases. Proceedings of the ACM, 1974.
6. Coffman, E.G. Jr.; Elphick, N.J.; and Shoshani, A. System Deadlocks. Computing Surveys, vol. 3, no. 2, June 1971.
7. Davenport, R.A. Distributed Database Technology - A Survey. Computer Networks, vol. 2, no. 3, July 1978.
8. Etchison, K.L. Early Deadlock Detection Mechanism. IBM Technical Disclosure Bulletin vol. 20, no. 8, p3063-3065, January 1978.
9. Etchison, K.L. Multiple-Owner Multiple-Level Deadlock Detection Mechanism for Resource Contention Control. IBM Technical Disclosure Bulletin vol. 20, no. 8, p3066-3069, January 1978.
10. Everest, G.C. Concurrent Update Control and Database Integrity. Data Base Management (Klimbie, J.W. and Koffeman, K.C., Eds.). North Holland Publishing Co., 1974.
11. Frailey, D.J. A Practical Approach to Managing Resources and Avoiding Deadlocks. Communications of the ACM vol. 6, no. 5, May 1973.
12. Haberman, A.N. Prevention of System Deadlock. Communications of the ACM vol. 12, no. 7, July 1969.
13. Hebalkar, P.G. and Tung, C. Logical Design Considerations for Distributed Data Base Systems. Proc. COMPSAC '77, November 1977.
14. Isloor, S.S. and Marsland, T.A. An Effective 'On-Line' Deadlock Detection Technique for Distributed Data Base Management Systems. Proc. COMPSAC '78 Software and Applications Conference, November 1978.

15. King, P.F. and Collymeyer, A.J. Data Base Sharings - An Efficient Mechanism for Supporting Concurrent Processes. AFIPS 42, 1973.
16. Lamport, L. Concurrent Reading and Writing. CACM vol. 20, no. 11, November 1977.
17. Macri, P.P. Deadlock Detection and Resolution in a CODASYL-Based Data Management System. Proceedings of 1976 ACM-SIGMOD International Conference on Management of Data.
18. Maryanski, F.J. A Survey of Developments in Distributed Data Base Management Systems. IEEE Computer, February 1978.
19. Peebles, R. and Manning, E. System Architecture for Distributed Data Management. IEEE Computer, January 1978.
20. Rosenkrantz, D.J. et al. A System Level Concurrency Control for Distributed Database Systems. Proc. of 2nd Berkeley Conference on Distributed Data Management and Computer Networks, May 1977.
21. Rosenkrantz, D.J.; Stearns, R.E.; and Lewis, P.M. System Level Concurrency Control for Distributed Data Base Systems. ACM Transactions on Data Base Systems, vol. 3, no. 2, June 1978.
22. Rothnie, J.B. and Goodman, N. A Survey of Research and Development in Distributed Database Management. Third International Conference on Very Large Databases, 1977.
23. Schlageter, G. Access Synchronization and Deadlock Analysis in Database Systems: An Implementation-Oriented Approach. Information Systems 1, 1975.
24. Stonebraker, M.R. and Neuhold, E. A Distributed Data Base Version of INGRES. Proceedings of Berkeley Workshop on Distributed Data Base Management and Computer Networks, May 1977.
25. Thomas, R.H. A Solution to the Update Problem for Multiple Copy Data Bases which uses Distributed Control. Bolt, Beranek and Newman Tech. Report BBN-3340, July 1976.
26. Thomas, R.H. A Majority Consensus Approach to Concurrency Control for Multiple Copy Data Bases. ACM Transactions on Database Systems, vol. 4, no. 2, June 1979.
27. Zemrowski, Kenneth M. Problems of Data Base Use in a Distributed Data Network. Tutorial on Distributed Processing, IEEE Computer Society (IEEE Catalog No. EHO-127-1) 1978.

V. DATA BASE SECURITY

This section deals with the issue of security as it relates to a distributed data base. Many of the security considerations are identical to those encountered in a centralized data base. Those considerations which are identical are, for the most part, not dealt with here. Instead, we concentrate on the considerations which arise solely or primarily because the data base is distributed.

Examples of considerations which are equivalent to those in a single, centralized data base include:

- (1) Physical Security - Protection against malevolent or accidental damage as a result of natural disaster (including fire) or the actions of individuals.
- (2) Communications Security - Protection against malevolent or accidental interruption of the automated communication path as the result of natural disaster (including fire) or the actions of individuals.
- (3) Data and Program Security - Protection against malevolent or accidental alteration or disclosure of data, schemas, or directories.

- (4) Environmental Security - Protection against malfunction of the system as the result of malevolent or accidental alteration of the environment, including power sources and the control of environmental temperature, humidity, and pollution.

Considerations which are unique to the distributed data base environment include:

- (1) Authorization - The distributed data base envisioned in this study has an "owner." The "owner" has the responsibility to determine which user(s) are authorized to add, delete, and change data base structures and the values of data elements.

In many respects, this problem is no different than in a single data base, but since segments of the data base may be located independently of the authority to modify them, there are special problems in the distributed data base.

- (2) Isolation - Within the constraints imposed on the model systems considered for this study, it is possible and likely that a node will possess a data processing facility which contains data from other, unrelated data bases and will be processing unrelated applications using these unrelated data bases, at the same time that the distributed data base is being processed.

Steps must be taken to isolate the data and the processes performed at a node in its own behalf from those in the distributed data base and vice-versa.

- (3) Uniformity - Uniformity of security procedures, practices, and safeguards must be imposed in the distributed system, to assure that each node is as secure as all other nodes, since the security of the overall system may be compromised by reduced security at a single node.

A large body of research data and policy papers has been produced on the subjects of security and privacy in the data base environment. Several are cited in the accompanying bibliography. However, very little research has been reported on the special problems which arise in the distributed data base.

5.1 THE GENERAL SECURITY PROBLEM

Generally, the need for security is not perceived as important by operating staff on a day-to-day basis. Checks and procedures may frequently be performed in a perfunctory manner in many computer installations.

Distributed processing and distributed data bases have increased security problems many-fold. The administrative and technical overhead of maintaining a geographically dispersed security system is many times that of a single location. This very process opens new areas for attack and new subjects for discussion.

Security of a distributed data base has been largely ignored by writers. It seems clear that the partition and replication of data and directories must have some impact on the security of the data; no one has chosen to address these questions other than by reference, e.g.,

"The security problem can be rendered more or less difficult in a distributed system than in a centralized one, depending on details of the system architecture and structure of the application" [10]

There are, however, a number of well discussed topics which have peculiar thrust and urgency for the designer of a secure distributed data base system, and there have been a number of areas mentioned as needing investigation which hinge directly on the characteristics of distributed data base systems. For example, the question of which processor should act on a given request has been treated often, but only in terms of performance criteria such as communication line loading, processor loading, and memory loading. Processor selection is, however, an important consideration when considering the security of a system. The distributed data base system must ensure that processes involving secure data are performed only on secure machines.

5.2 RELEVANT SECURITY TECHNIQUES

There are three important components in any security system: access control; use control; and surveillance. Access control may be effected through physical or procedural means; a user may, for instance, need a building pass and a key to gain access to computer equipment and may need passwords or other identifications to establish a computer interface with the data base. Use control is performed by the software; it prevents an identified person from performing operations on the data which have not been authorized for the user. Surveillance is constant vigilance to detect attempts to break security and to find and stop those security breaches which will occur.

Even though security systems may involve much technology, most of them are administered by people and depend for their success on the attitude of those using or controlling the system. All too often security systems fail due to the complacency of those who use it.

There appears to be an inverse relation between the availability of data and the ease of maintaining security. Tactical data systems present a unique challenge to the designer. Not only must he deal with problems of data unavailability due to equipment failure but those due to covert enemy action. In tactical systems data must be most available just when it needs to be most secure.

The following paragraphs discuss some design considerations that apply to distributed data base systems.

5.2.1 Telecommunications

Distributed processing and distributed data bases have come into use only recently as the cost of hardware has dropped in relation to the cost of data communications. It is ironic then that the telecommunications links represent the single most serious threat to the integrity of the security of a distributed data network.

The uninformed consider the task to wire tapping difficult and the extraction of data as so complicated as to require arcane skills and vast amounts of equipment. Effective wire tapping is easy and the extraction of data is no great task. Communications lines are as a rule hidden but accessible. The attachment of tapping leads takes but a moment and can be done in such a way that the untrained eye cannot detect their presence. Faced with the virtual guarantee that communication lines will be tapped, the system designer must seek means of denying intelligence to the would-be listener or making the listener's task so complicated it becomes not worth the effort. Typical efforts in this direction are:

- High speed lines
- Multiple routing
- Multiplexing
- Encryption.

5.2.1.1 High Speed Lines

While usually not a sufficient obstacle to prevent the gathering of information, a high data rate, in lines with little idle time, can make it more difficult to insert spurious messages. The use of high speed lines will enhance the effectiveness of other anti-tap measures.

5.2.1.2 Multiple Routing

Successive messages bound for the same destination need not be sent over the same communication line. By sending successive messages on different lines, according to some prearranged pattern, the system presents considerable obstacles to any agency who would listen-in or insert queries or false messages. The closer the pattern approaches "randomness," the more difficult it will be to analyze. Random here, of course, means from the viewpoint of an external observer; even a "random" pattern must be known in detail by sender and receiver. Even the most "random" pattern will be analyzed in time, and there is a maximum interval beyond which one pattern should not be used. This interval is a function of the length of the pattern and the message rate.

5.2.1.3 Multiplexing

By sending portions of a message via different communication lines, the secure system greatly increases the workload of the intelligence gatherer by a factor equal to the number of communication lines devoted to sending one message. The picture can be farther complicated by altering, again in some "random" fashion, the lines over which various portions of the message will be sent.

5.2.1.4 Encryption

Encryption has been traditionally handled via substitution algorithms, which are very amenable to analysis, or substitution tables, which consume great amounts of processor time and will eventually be broken anyway. Encryption has in the past been dismissed as valueless. Recently, the National Bureau of Standards has issued a Data Encryption Standard (DES) which is based on the public-key system. The hardware to implement DES is becoming available.

With DES, the user selected decrypting key is manipulated using discontinuous mathematical functions (notably modular arithmetic) to generate an encrypting key. Due to the discontinuity in the generation of the encrypting key, the decrypting key cannot be derived from it. The encrypting key can be published for use by anyone wishing to send messages to the key owner. Since discontinuous functions are again used in the application of the encrypting key, it is not possible to derive the contents of the encrypted message, even though the encrypting key is known. At least one commentator [9] feels that encryption is the only acceptable way to secure communication lines.

Of the four data link securing stratagems suggested, encryption, in accordance with DES, is by far the best. Multirouting and/or multiplexing messages impose significant burdens upon the wire tapper, but the expense of the necessary additional communication lines tends to offset the value of the relatively cheaper hardware.

5.2.2 Operating System

In the absence of hardware intensive solutions, there are software protections against modification of data, insertion of false messages, and unauthorized users:

5.2.2.1 Password/Algorithm

Passwords and sign-on algorithms have long been used in computer security. Given the off-hand attitude most users have towards such things, the basic effect of using passwords is to provide a false sense of security. Unless vigorous procedural standards are followed by all users of the system, passwords and sign-on algorithms cannot be counted on to stop any but the casual interloper.

5.2.2.2 Operator/Terminal Relation

One way to increase the effectiveness of passwords and sign-on algorithms is to have the computer system maintain information on time-of-use and user-terminal identification. By limiting the hours and the equipment for which a sign-on is allowed, the system designer can discourage the unauthorized use of specific access codes.

5.2.2.3 Positive Acknowledgment

Positive message acknowledgment serves a twofold purpose. It provides a check on the proper operation of the communications network and makes it difficult to delete or insert messages, via wiretap, without detection. Each message is tagged with a serial number, representing the position of the message in the session. Positive acknowledgment is expected back, containing the identification of the previously transmitted message. When acknowledgment to an unanticipated message number, or an unexpected message number itself is received, operator notification or link abandonment can occur as seems appropriate.

5.2.2.4 Positive Operator Identification

Devices exist which can record the significant factors in signatures, fingerprint, hand geometry, and voice. These devices do not require separable or losable devices such as keys or compromisable information as passwords. While these devices have good records for keeping out unauthorized persons, they do occasionally reject authorized persons. The fault lies in the person; people are too changeable. The false rejection rate is low enough to be operationally acceptable, but user acceptance usually suffers as a result of a false rejection.

Attention has already been paid to various methods of validating appropriate users of communications networks. Several additional issues are raised in distributed data base systems. If the data or directory is centralized, a degree of security can be maintained by validating a user to a particular processor. Alternatively, each processor can validate each user either at sign-on or as the processor is called upon to provide data. The first solution, depending as it does on centralization, poses a serious threat to data availability whereas the second involves a great duplication of effort.

5.2.3 Data Base Management Systems

The DBMS protects data against not only active prying but accidental disclosure. It is recognized that the security of the DBMS is dependent on the security of the operating system in which the DBMS is associated. In

addition to such operational security techniques as passwords, etc., DBMSs usually employ a number of design stratagems.

The DBMS prevents casual and accidental access to the data base in conjunction with protections provided by the operating system. It does this because all references to data are handled through the DBMS schema. (The schema is a description of the physical and logical relationships of the various data items. It is usually unknown to the average user.) The unauthorized user must discover the complex logical data relationships of the schema if he is to use the data base outside the control of the DBMS. The sheer bulk of data which must be sifted discourages the casual tinkerer.

The authorized user is isolated even farther from the data through the use of subschema, which is a logical description of those parts of the data base to which the user or application programs has been granted access. Thus security considerations have been removed from application areas and placed in the hands of the Data Base Administrator. (This is based on the perhaps naive assumption that the Data Base Administrator maintains control of the allocation of subschema.)

The granularity of the data—that is, the size of the smallest referenceable data unit—is important even where subschema are employed. If the program receives an entire data record, even though it requires only a few nonsecure data items, the integrity of the security system could be compromised. Many commercially available DBMSs provide a grain size of "data items." Even where a granularity greater than one data item is required due to DBMS constraints, the integrity of the system can be maintained by partitioning the data into secure and nonsecure grains.

5.3 ASSESSMENT OF CURRENT TECHNOLOGY

In spite of the questions and dangers herein discussed, many areas of the security function in distributed data base systems are well understood. Though the technology for wiretapping is easily grasped and wiretap itself difficult to prevent, the tools for detecting such intrusion are available and lack only the will to use them. The problem of site security is no different from that of any other data processing installation. A number of devices and

stratagems are available to control access to secure systems. The principal danger to the security of distributed data base systems is due to a lack of understanding of inherent weaknesses of the distributed data base per se.

5.4 AREAS FOR FURTHER STUDY

There is no aspect of distributed data base security which could not do with more analysis and discussion.

A number of unique characteristics have been identified by investigators as important to the security of such systems. The distinguishing attribute of a distributed data base is the partitioning of the data base among a number of sites. This factor has been mentioned as central to the security problem, but no analysis has been done. Replication of data can serve to reduce or increase security problems, depending on the environment. An analysis of the loss, monetary or strategic, due to unavailability must be made as a prerequisite to intelligent plans. No tools have been identified to aid in this analysis.

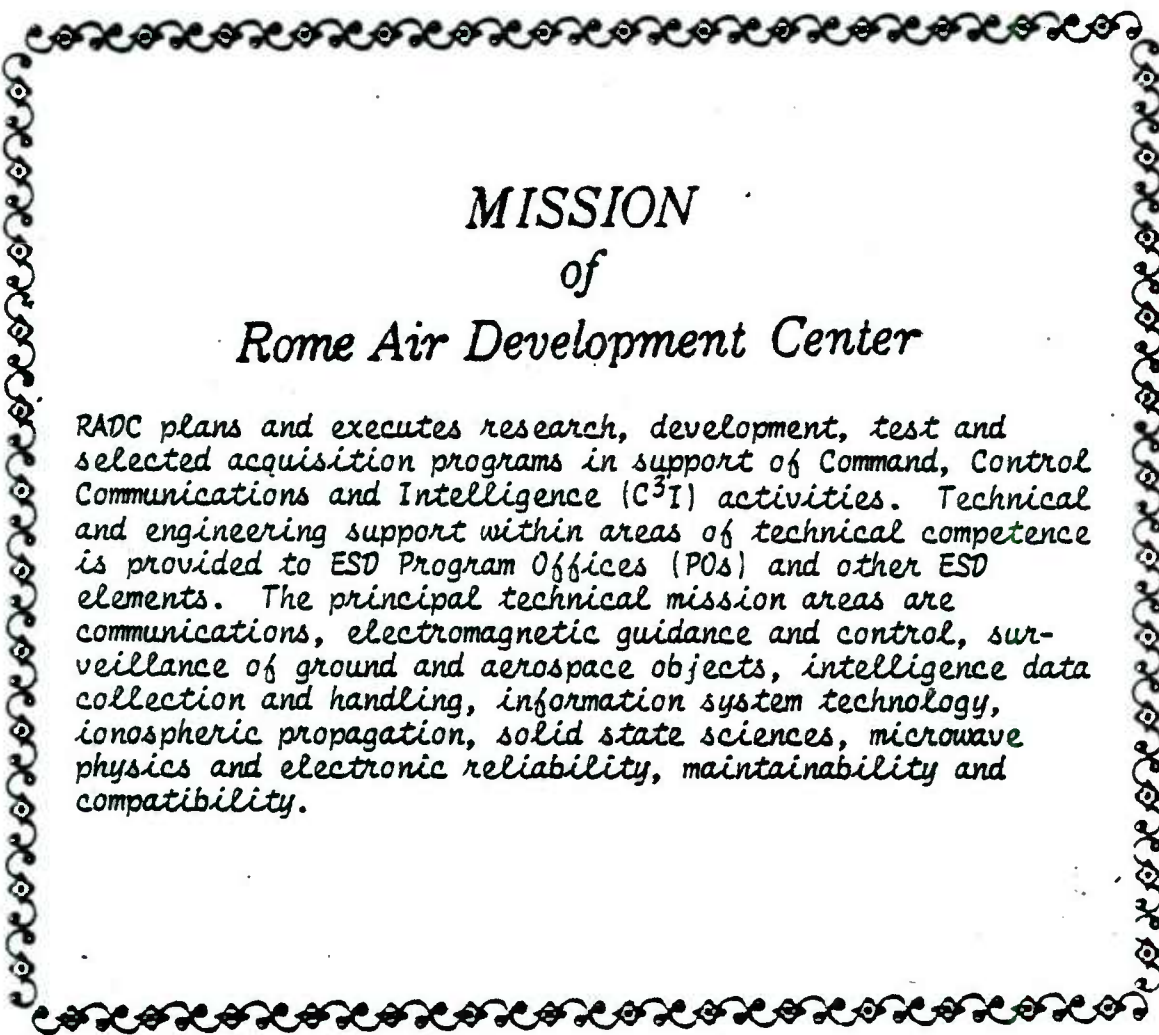
Another question which has been asked deals with the selection of a processor to handle manipulation of secure data. A site requesting data may not, from the security standpoint, be the best for processing the raw data. If the distributed data base system is heterogeneous for the operating system, the processor site selection can become very important.

The security system itself is affected by distribution. The more distributed the security system becomes, the more vulnerable to examination and subversion; yet, if security is centralized, the problem of data unavailability arises, as does the risk of penetrating the telecommunication system.

Several areas have been mentioned as needing further study. These areas cannot be considered individually. For example, the replication of part or all of the data base has effects on the selection of the processor and the desirability of having a distributed security system.

BIBLIOGRAPHY

1. Abrams, M.D.; Branstad, D.K.; Browne, P.S.; and Cotton, I.W. Computer Security and Integrity. IEEE Computer Society, 1977.
2. Aschim, F. Data Base Networks - An Overview. Management Informatics vol. 3, no. 1, February 1974.
3. AUERBACH Publishers, Inc. Telecommunications Security and Countermeasures, 1979.
4. AUERBACH Publishers, Inc. Security and Reliability: Design and Operational Concepts, 1976.
5. Davenport, R.A. Distributed Data Base Technology: A Survey. Computer Networks, vol. 2, no. 3, July 1978.
6. Deppe, M.E. and Fry, J.P. Distributed Data Bases: A Summary of Research. Computer Networks, vol. 1, no. 2, November 1976.
7. Downs, D. and Popech, G.J. A Kernel Design for a Secure Database Management System. Proceedings of a Conference on Very Large Databases, October 1977.
8. Fry, James P. and Maurer, John. Operational and Technological Issues in Distributed Data Bases. Data Communications Management. AUERBACH Publishers, Inc., 1978.
9. Maryanski, F.J. A Survey of Developments in Distributed Data Base Management Systems. IEEE Computer, February 1978.
10. Peebles, Richard and Manning, Eric. System Architecture for Distributed Data Base Management. IEEE Computer, January 1978.
11. Reiss, S.P. Security in Data Bases: A Combinatorial Study. Journal of the ACM vol. 26, no. 1, January 1949.
12. Rothnie, J.B. and Goodman, N. An Overview of the Preliminary Design of SDD-1: A System for Distributed Data Base. Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.
13. Sykes, D.J. Positive Personal Identification. Datamation, November 1978.
14. Ware, Willis H. Handling Personal Data. Datamation, October 1977.



MISSION of Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.