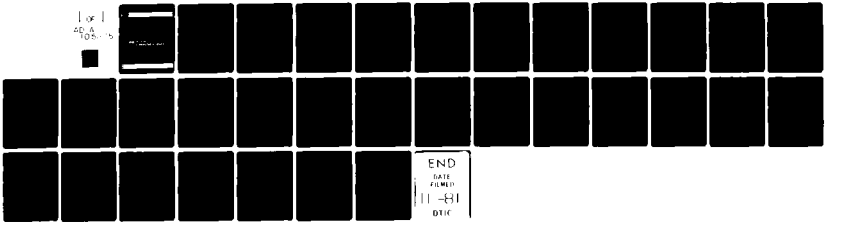
AD-A105 825 KANSAS STATE UNIV MANHATTAN DEPT OF COMPUTER SCIENCE F/G 9/2
DATA ACCESS IN DISTRIBUTED DATA BASE MANAGEMENT SYSTEMS,(U)
SEP 78 F J MARYANSKIL, P S FISHER DAAG29-78-G-0018
NL

UNCLASSIFIED



END
DATE FILMED
11-81
DTIC

Data Access in Distributed Data Base Management Systems

Fred J. Maryanski[1,2]
Paul S. Fisher[1]
Virgil E. Wallentine[3]

Sept 1978 33

KSU KANSAS STATE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

DAAG 29-78-G-0212
DAAG 29-78-G-0244

Data Access in Distributed
Data Base Management Systems

Fred J. Maryanskil,2
Paul S. Fisher[1]
Virgil E. Wallentine[3]

Computer Science Department
Kansas State Univerisity
Manhattan, KS 66506

TECHNIQUES

[1]The work of these authors was partially supported by U.S. Army Research Office Grant No. DAAG 29-78-G-0018.

[2]Author's address: Digital Equipment Corp.; 146 Main St.; Maynard, MA 01754

[3]The work of this author was partially supported by the Army Institute for Research in Management, Information, and Computer Systems (AIRMICS) and monitored by the U.S. Army Research Office Grant No. DAAG29-78-G-0200. To be published in the Journal of Information and Management.

I. Introduction

Distributed data base systems have been advocated as the solution to a large number of data processing problems by increasing data accessibility, security, and throughput while reducing cost and resource requirements. Unfortunately, commercially available distributed data base systems have not yet appeared. This paper attempts to provide the potential user or designer of a distributed data base system with an understanding of the basic operational characteristics of such systems. The emphasis is upon the mechanism for data access which is an essential component of any data base system. Our intention is that the reader gain an appreciation of the capabilities and complexities of distributed data base management from the explanation of the data access mechanism.

This paper first discusses the basic structure of distributed data base systems by detailing the functions of the system components. Then in parts three and four, mechanisms are presented for the placement and access of data in a distributed data base system. The fifth part deals with the movement of data among machines and then the sixth section briefly discusses the concept of multiprocessor backend machines. The final portion discusses data integrity considerations in distributed data bases.

II. Background

A distributed DBMS involves a network of computers whose software systems and applications share the functions of the data base system and the stored data. The processors in the network may be classified into any of the following four functional categories:

1. Frontend--to act as a telecommunications monitor and provide the user interface to receive input and transmit output;
2. Host--to execute the application program;
3. Backend--to control and provide data access through execution of data base system operations;
4. Bi-functional--the combination of host and backend functions.

The organization of distributed data base systems has been discussed by several authors [11,16,20,25]. While terminology in the various reports varies, their basic structures are very similar. The architecture of distributed data base management described in this paper is further explained in [18].

The most elementary form of an example of a distributed data base management system is the backend DBMS; see Figure 1. Canaday et al. [2] first proposed and prototyped the backend DBMS concept. A key component of any distributed data base system is the interprocessor communication system. The type of functions which must be supported by an interprocessor communication system are--

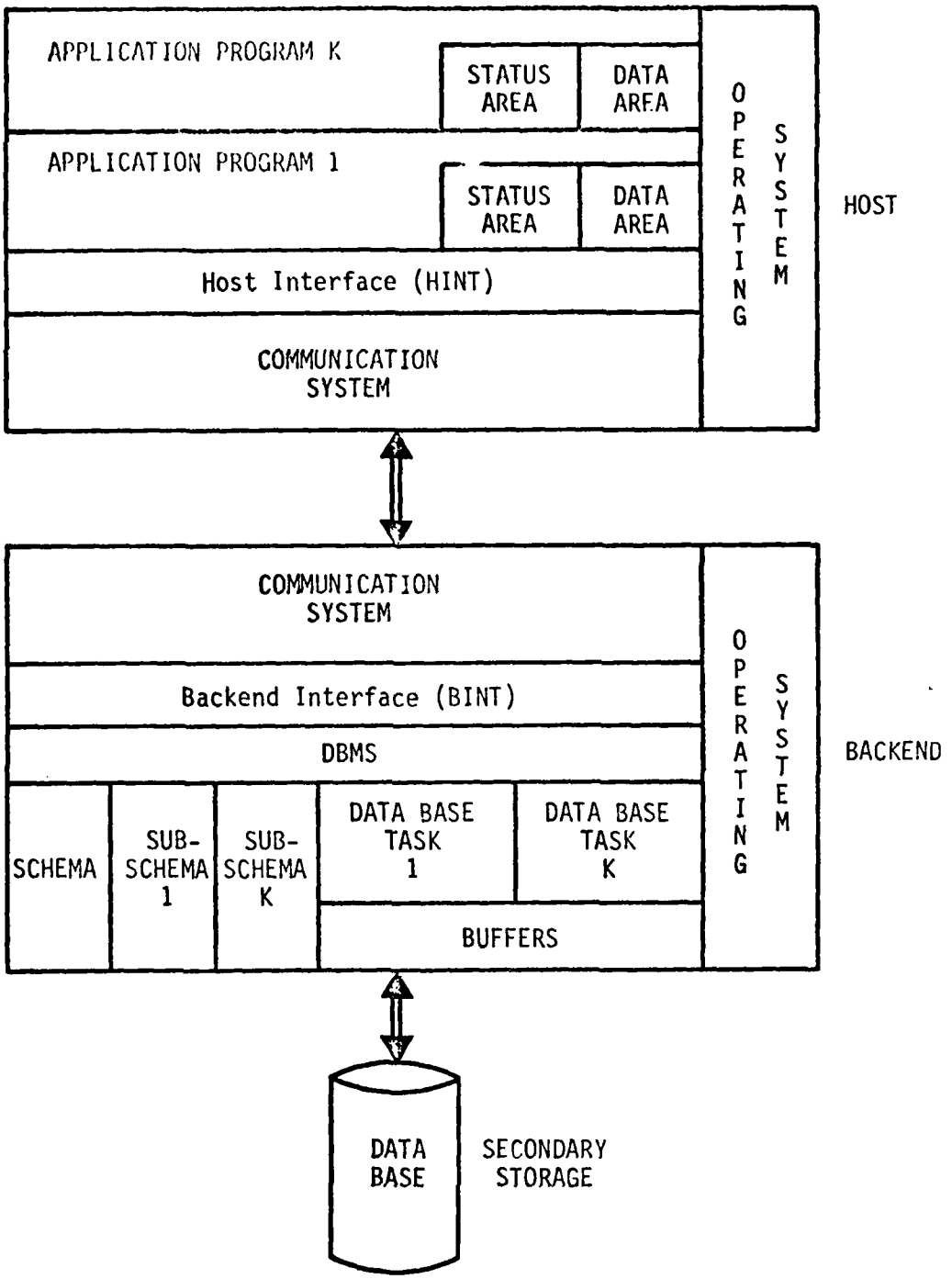


Figure 1

Software Distribution in a
Host and Backend System

1. transmiting data and commands between tasks (e.g., application program and DBMS tasks);
2. making the topology of the network transparent to the application program;
3. synchronizing the tasks which exchange data to insure no data is lost, garbled, or pilfered.

References [4,12,22,23,29] describe currently operational systems.

The utilization of an interprocessor communications system can be illustrated by considering the simple case of a single host, single backend system. A task in the host requests a service from the DBMS system. The residual DBMS system contained on the host interacts (through the communications system) to request the service from the backend. A task on the backend computer performs the data base function, including any necessary I/O operations on the data base. The results of that activity are then returned to the requesting host task (through the communications system). Hence, the backend processor can be thought of as an I/O device for the host machine.

Backend data base management systems have been shown to be feasible approaches to the enhancement of data processing systems [2,14,18]. Further, it has been suggested that a backend data base system minimizes the task-switching overhead on the host CPU and provides for a smaller primary memory requirement for both data base system and application software on the host CPU in comparison to a single machine DBMS. In general, a backend DBMS, due to its capacity for concurrent execution, has the potential for providing a

substantial increase in throughput over a single machine system.

Figure 1 illustrates the distribution of software functions over the two processors in a backend-host DBMS environment. As depicted in the figure, the data base software in the host consists only of a host interface routine (HINT) between the Communication System and the application program. For each data base command issued by the application program, the HINT formats a message which indicates the action that the backend processor must take to complete the data base operation. The message is transmitted to the backend machine via the Communication System. The backend interface task (BINT) unpacks the message and passes the information to a Data Base task which acts as a surrogate at the backend node for the application program. The Data Base task presents the data command to the DBMS model. The execution of the command may result in one or more I/O operations. Completion of the data base operation results in data and/or status information being returned to the application program; the path for access and response is indicated in Figure 2.

The backend DBMS configuration can be extended in a variety of ways in realizing a distributed data base. A first step might be to spread the data base management function over several backend processors. The primary difficulty in the development of a multiple backend configuration is the determination of the correct backend processor for a particular data base operation. A mechanism for the selection of a backend processor is presented in

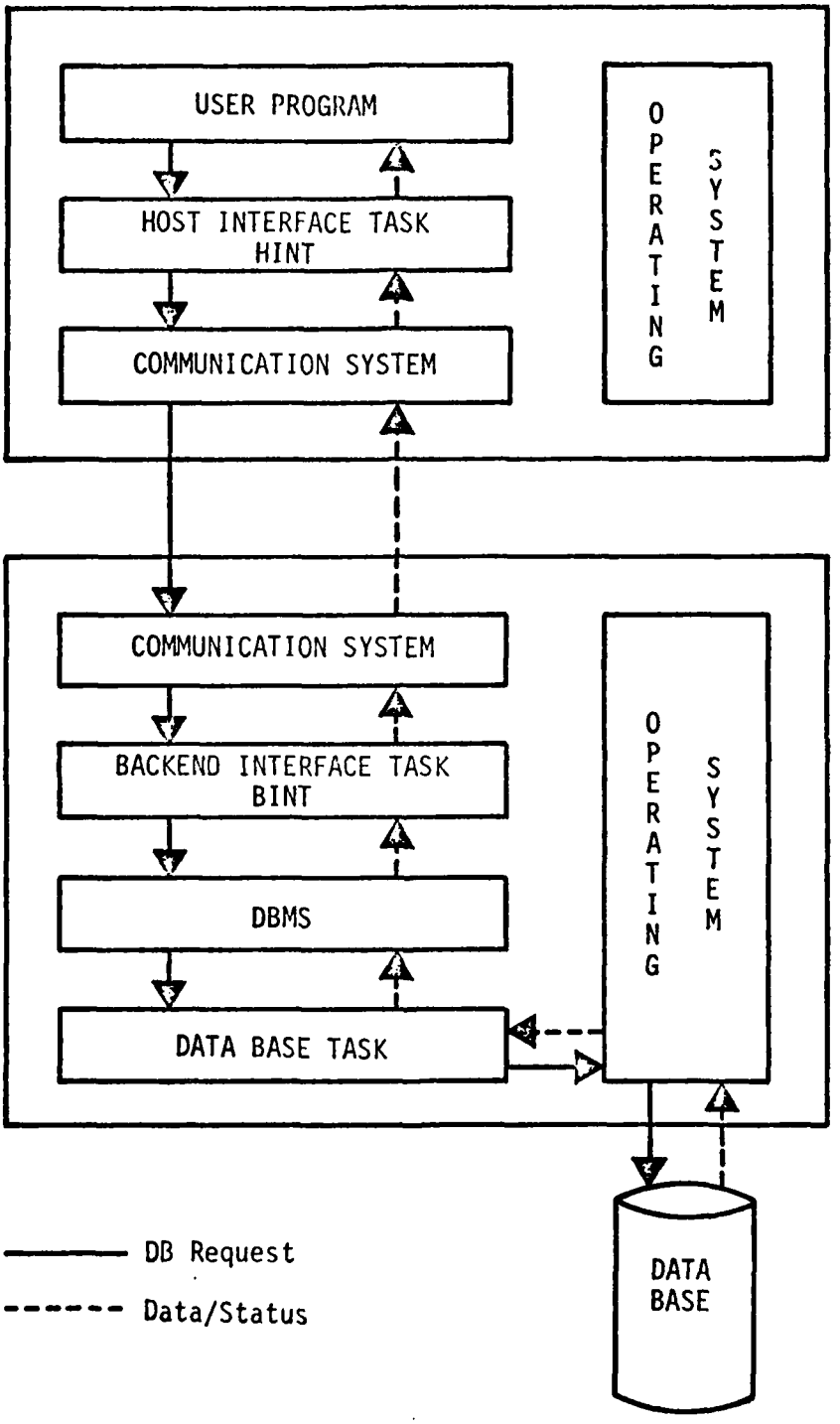


Figure 2

Information Flow in Backend DBMS

Section IV.

A distributed data base network may contain several hosts, as well as several backend processors. In this environment, each host processor selects the appropriate backend machine to execute the data base command and communicates with that machine in the manner described previously. Return of data and status information by the backend processor is carried out by attaching the identifier of the host processor to the transmitted data.

The final step in the evolution of distributed data base network topologies is to combine the functions of a host and a backend machine on a single processor to form a bifunctional machine. Use of bifunctional machines in a distributed DBMS provides the potential for more efficient utilization of system resources. The only restriction as to the function of a processor in a distributed DBMS should be its physical connections to secondary storage. Figure 3 presents a data base network with host, backend, and bifunctional machines.

There are problems associated with the distributed DBMS system; e.g., many of the requirements (logging, recovery, deadlock) found in single machine DBMS's become considerably more complicated in a distributed environment. In addition, some problems (e.g., data translation, communication, data redundancy and allocation) also arise. However, the potential benefits to be derived from a distributed data base system have spurred considerable research activity in this area [11,16,20,26].

Implementation of distributed data base systems with

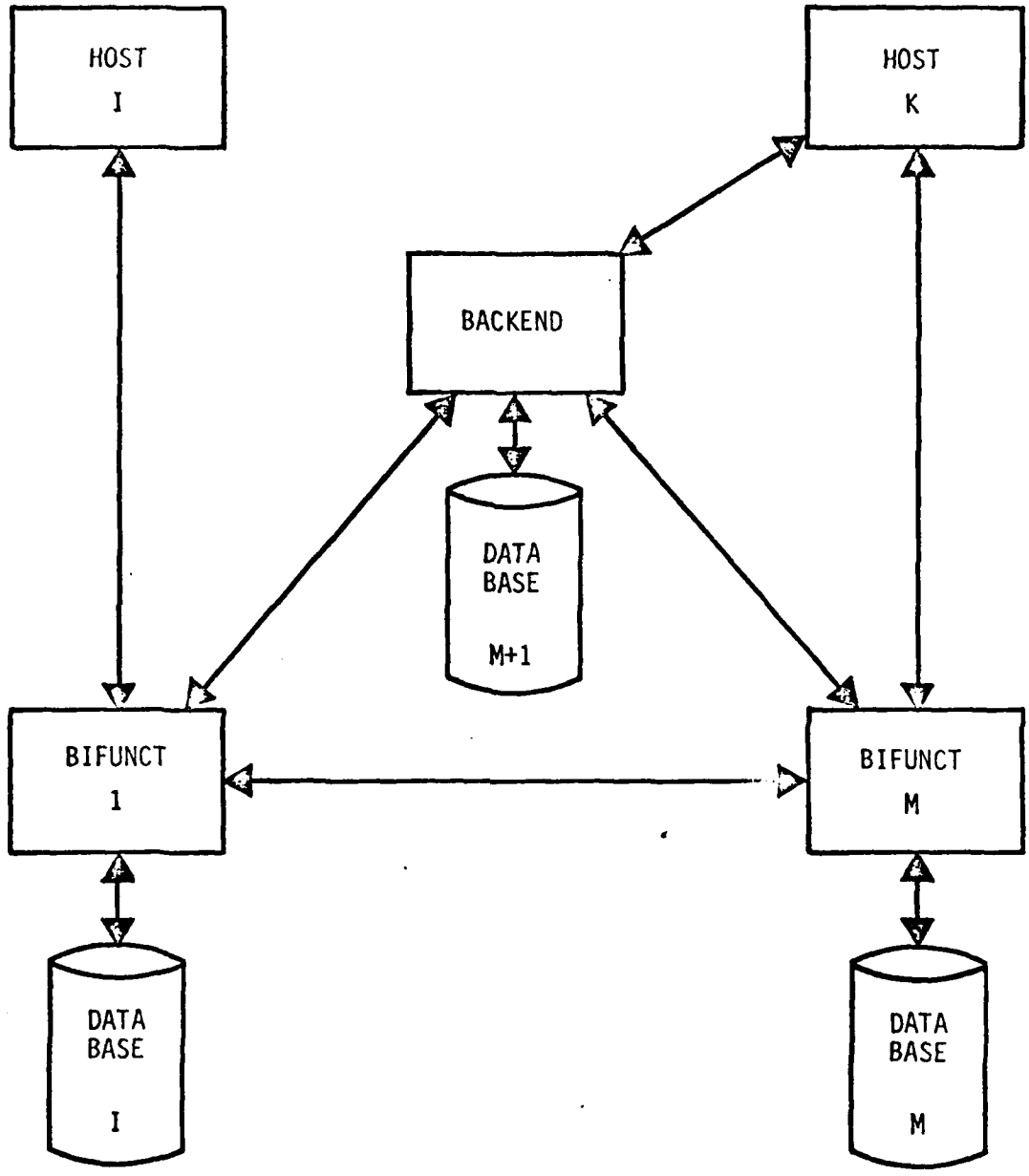


Figure 3

Distributed DBMS

more than two processors has been limited since attention has focused upon basic problems of backend software development, intertask communication, and performance optimization, as well as concurrent updating encountered in a distributed DBMS. Such problems could cause major performance problems in the operation of a distributed DBMS. Theoretical solutions to the concurrency control problems have been suggested, but performance has not yet been properly tested [1,6,7,25,26,28]. Allocation of files in distributed data base systems using linear programming models has also been studied [3,5,9,13,15,21]. The problems of integrity and security in distributed data bases and recovery have received a limited amount of attention [8,17].

The problem of data placement and access requires a solution before a distributed data base system can be developed. As stated previously, the host interface must be able to identify the proper backend processor for each data base operation. In addition, there must be a mechanism for specifying the location of data in the network. An additional requirement of a data access mechanism is that procedures must exist for the movement of data among network nodes under control of a (utility) program or an operator. This paper presents a methodology for data placement and access in a distributed DBMS satisfying this particular set of requirements.

III. Data Placement

A host interface routine must have the ability to determine the location of a backend node containing data requested by an application program. Alternative methods for obtaining this information are to broadcast a query to all backend nodes or to utilize a directory lookup facility. We advocate the use of a directory since it involves less communication. For purposes of this discussion, the directory will be termed the network data directory.

A key concept of the network data directory, as viewed by the authors, is that it does not contain the specific location of each data item in the data base; but rather that from its contents the location of every data item can be derived. For a data base management system, a unit of data distribution (granule) must be selected so that there is a one-to-one mapping from the granule to the backend machine. The actual distribution unit is dependent upon the organization of the DBMS and will vary with the granule equivalency available within the actual DBMS utilized [8,10,24].

The language for the definition and manipulation of the network data directory can exist as a stand alone facility for use by the DBA, or it can be incorporated into the data definition language of the data base system. In order to specify the placement and access of a granule in a distributed data base, the backend processor controlling access to the granule must be named in addition to the host processors which may execute application programs accessing that granule. This information can be compiled from

statements similar to those shown in Figure 4 into the network data directory format depicted in Figure 5.

The placement of the network data dictionary within the network of distributed machines is a trade-off of performance, reliability, and complexity of maintenance. A central directory can be maintained easily, but a high penalty in performance is paid to reference it across the network. Furthermore, if the machine holding the directory fails, the entire data base system fails. A distributed directory can be composed which places a copy of the directory entries for those granules accessible by a particular host at each host node. The performance improvement is quite clear since the directory access is within the host running the accessing program.

The listing of the host processors in the network data directory provides additional security for the distributed DBMS. That is, the DBA can specify whether the granule can be accessed globally or whether is must be restricted to certain application functions residing on specific host machines. Security restraints are imposed by the host access control lists to limit the access to a granule by specifically designated application programs. File names are maintained in the network data directory to facilitate the movement of files among machines in the network in the case of failure or to improve performance. Figure 6 pictures a distributed data base and its network data directory.

The utility of the host access control list (see Figure 4) is to validate dynamically each access of a host

GRANULE IS identifier1

BACKEND IS identifier2

DEVICES ARE identifier3{,identifier4,...}

HOST-TASKS ARE [(identifier5,access rights) / {(identifier6,access rights),...} / ALL]*

Legend:

* Host Access Control List

Figure 4

Network Data Directory Definition Statements

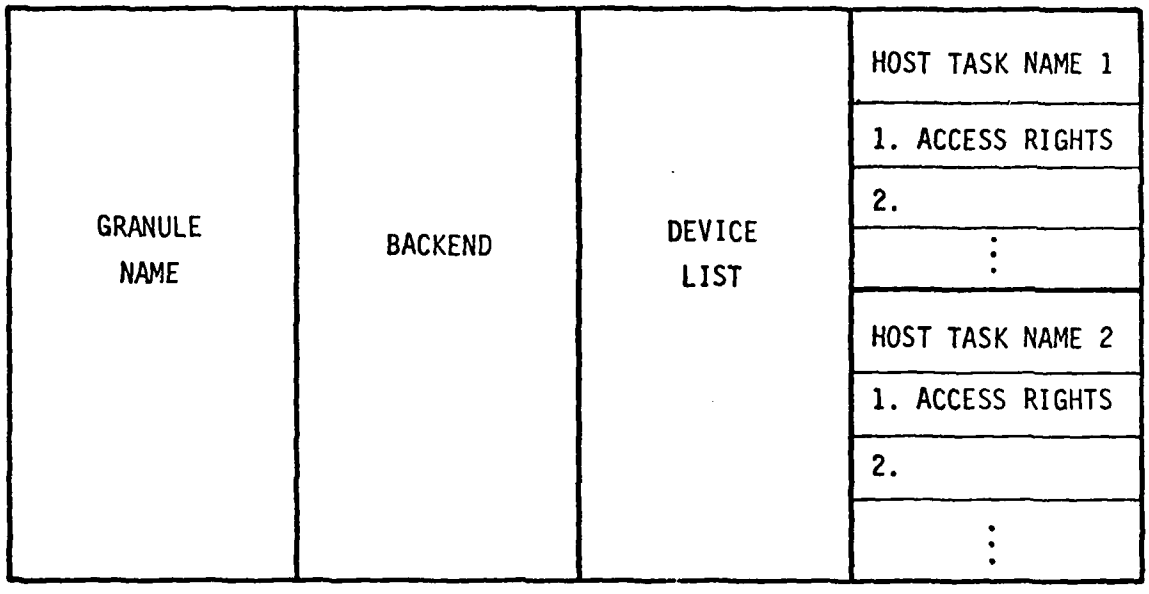
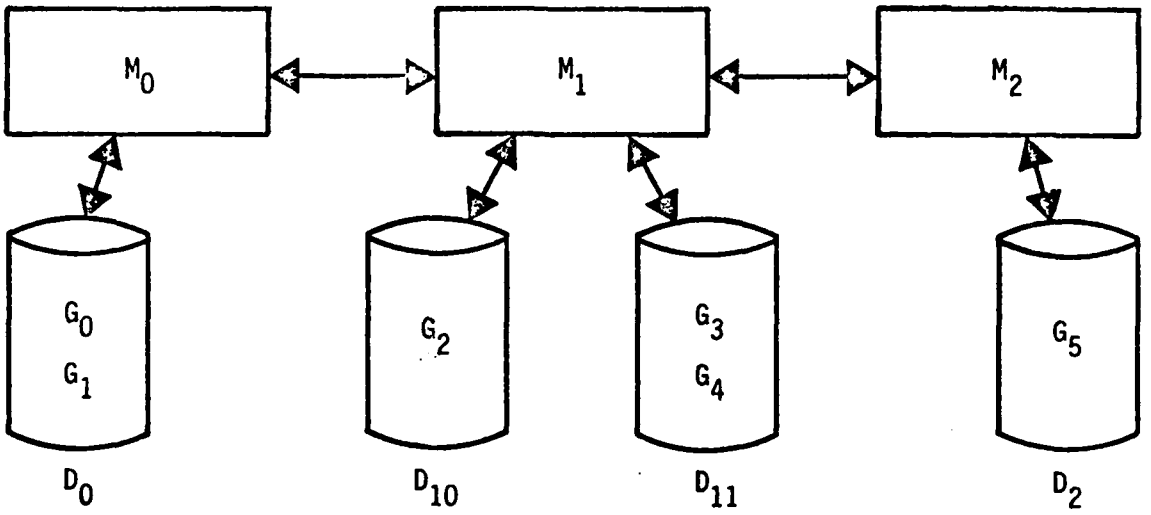


Figure 5

Format of Network Data Directory

Data Base (M_K-Machine, G_K-Granule, D_K-Device)



Network Data Directory

Granule	Backend	Devices	Hosts
G_0	M_0	D_0	M_0M_2
G_1	M_0	D_0	ALL
G_2	M_1	D_{10}	M_1
G_3	M_1	D_{11}	M_0M_2
G_4	M_1	D_{11}	M_2
G_5	M_2	D_2	ALL

Figure 6

Sample Network Data Directory

application program to a granule. The access control list is distributed to the backend at the time of execution of the network data directory definition statements. This list contains the identification of each program which can access the granule and the manner of permitted access. At the same time, a capability list (i.e., a list of granules accessible by a host and the manner of access of each) is constructed and sent to the hosts. This permits checking of access rights at both the host and backend. This technique provides dynamic verification of decisions which prevents most errors on either host or backend from permitting unwarranted data access, and thus individual errors must be complementary to permit erroneous access to the data.

IV. Data Access

There are three phases of data access during the execution of an application program. First, all access rights to data are established. Once this has been checked, the program issues data manipulation commands. Third, the termination of the program data request must be indicated.

In order to establish the application program access rights, the application must issue a command, or commands, indicating the granules that may be accessed. For example, if a CODASYL DBMS were operating in a distributed environment, a READY AREA statement could serve this purpose. The host interface routine (HINT) extracts the granule names from the access request command and, using the name as a key to the network data directory, determines the

backend machine for each granule. A message is then transmitted (via the communication facility) to the backend interface (BINT) requesting that the granule be made available to the application program and that a task be created, if one does not already exist, to execute the data manipulation commands.

Several problems may arise which can hinder the establishment of access rights. If the application program does not satisfy all integrity and security requirements, its request will be rejected; but even when integrity and security constraints are met, device and machine availability problems could arise. If after receiving the access request from BINT, the DBMS discovers that the files for the granule are not on-line, it must request that the backend operator mount the data file. If the requested file can be made available, access to the granule is granted by the DBMS to the application program and an appropriate response is returned to the application program; otherwise, the application program is notified of the unavailability of the granule.

In many data base systems, a single data base command may result in operations on data units in several granules. For example, in a CODASYL-type DBMS, the deletion of an owner record from the data base may result in the deletion of its member records. In such situations, the BINT routine on the backend machine executing the data base command must determine the granule names for each data unit involved in the operation. The physical location of each granule is determined from the network data directory. If other

backend machines control access to granules affected by this command, messages are transmitted to them by BINT indicating the operations to be performed. This process could reoccur several times before completion of the data manipulation command, depending upon the complexity and distribution of the data base. When the BINT routine on the original backend machine has received completion messages from all of the other BINT routines engaged in the completion of the command, it then transmits a message with the appropriate data and status information to the HINT routine on the host machine. HINT then passes this information to the application program.

V. Data Movement

In a distributed DBMS, it may be desirable for reasons of efficiency or security to change the physical location of data or programs. Movement of application programs is certainly a function of a network operating or load leveling system. The only data base involvement is setting the network data directory (either statically or dynamically). Movement of data is done on a granule basis and can occur either by a programmed transfer between storage devices of different backend machines or by an operator physically moving a storage device volume from one computer to another.

The case of data movement under control of an application, or utility, program will be considered first. When this occurs, the network data directory must be

automatically updated to reflect the new location of the transferred granules. There will be no effect on any host application program. The data transfer requires that all activity on the granules cease during movement. In order to force a quiescent state, all new data base requests for the granules must be held. Once all pending data base requests are complete for the granules, the data transfer can be accomplished (using network utility programs). The procedure is:

Procedure 1

Let B_s be the sending backend machine, B_r be the receiving backend machine, U_s the network operating system utility on B_s, and U_r be the network operating system utility on B_r. The steps indicated below must be followed to move a granule, G, from B_s to B_r.

1. The U_s routine is called and instructed to move G to B_r.
2. U_s notifies the BINT routine on B_s that G will be moved.
3. BINT on B_s will not accept any further messages for data base operations on G from the message system routines on B_s. Any such pending messages are returned to their host machines.
4. The BINT routine on B_s determines the host processors for G from the network data directory.
5. The BINT routine on B_s requests that the message system on B_s instruct the message utilities on

all host machines for G that any messages for this granule are to be held in the host machine by the message system.

6. The BINT routine on B_s will notify U_s when all data base operations on G are complete.
7. U_s sends a message to U_r indicating that G is to be transferred to B_r.
8. U_s writes G onto secondary storage, thus insuring that the latest version of G will be moved.
9. Upon completion of the transfer, U_r informs all HINT and BINT routines to change the network data directory entries for A from B_s to B_r.
10. The BINT routine on B_r obtains the list of hosts for G from the network data directory.
11. The BINT routine on B_r notifies the host message utilities that any queued requests for data base operations on G may now be sent to B_r.

The interaction of the utility routine, host and backend interface, and system software resulting from the execution of Procedure 1 is illustrated in Figure 7. Only the software directly involved in the data transfer is pictured. Figure 8 give the network data directory for the sample configuration of Figure 6 if G_4 were transferred to D_2 on M_2 from D_{11} on M_1.

The situation in which a physical file is moved manually is more complex than a programmed transfer. The most difficult situation arises when the file is active on a

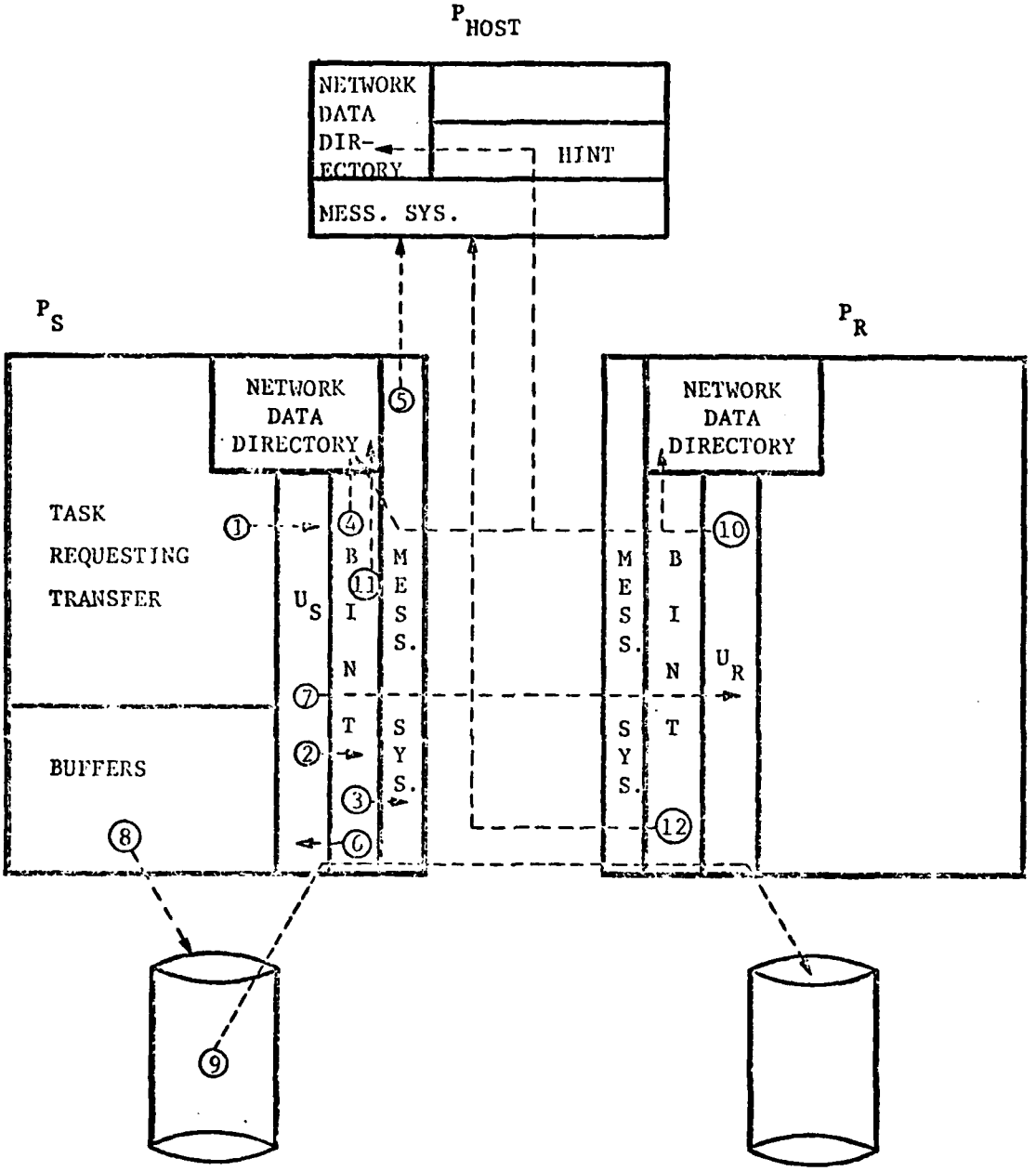
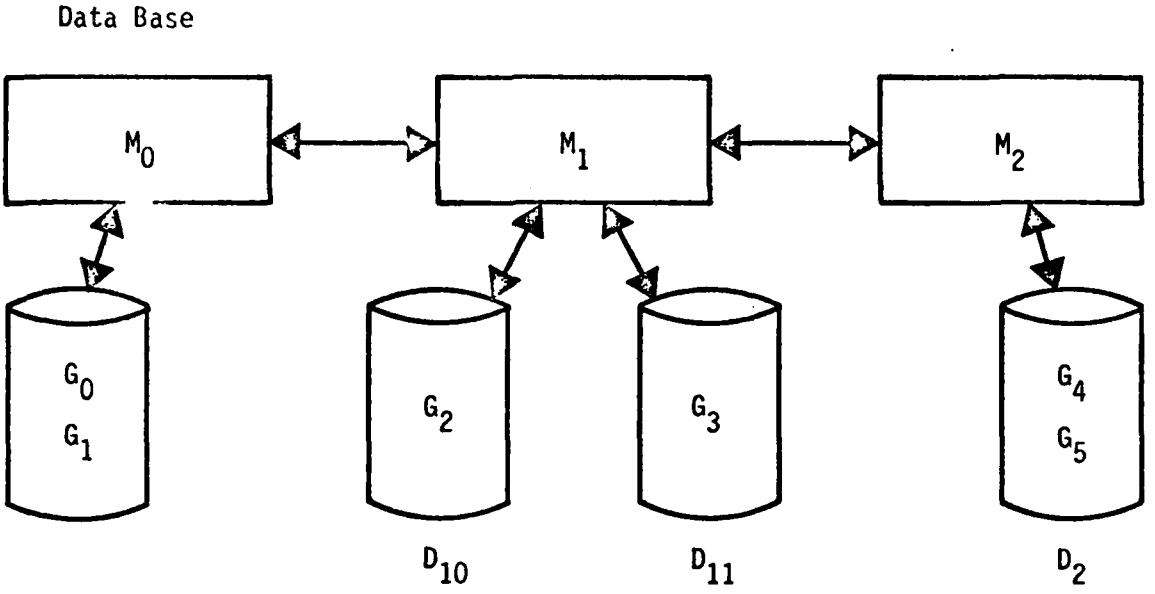


Figure 7

Programmed Data Transfer



Network Data Directory

Granule	Backend	Devices	Hosts
G_0	M_0	D_0	M_0M_2
G_1	M_0	D_0	ALL
G_2	M_1	D_{10}	M_1
G_3	M_1	D_{11}	M_0M_2
G_4	M_2	D_2	M_2
G_5	M_2	D_2	ALL

Figure 8

Sample Network Data Directory After Data Movement

backend machine and the operator instructs the operating system to remove the file from on-line status. As part of the procedure for making the file unavailable, the list of granules on the affected file is obtained from the device header list. BINT is instructed to manage the cessation of all activity for each of the designated granules by using the method described for programmed transfer of the granules. Any pending requests for operations on data in the granule will be held in the host processor. When BINT determines that all activity has terminated for each granule on the file, the operating system informs the operator that the file may be removed from the backend machine. When the file is mounted on the new backend machine, its presence must be made known to the network. The BINT routine on the backend machine receiving the file must instruct all other interface routines that the network data directory has been modified to reflect the new location of the granules. The BINT routine then communicates with the HINT routines of each transferred granule indicating that it will now accept data base operations for those granules.

Procedure 2 describes the operations required in the physical movement of a file between backend machines in a distributed data base system with no redundant data.

Procedure 2

Let B_s and B_r be backend processors as in Procedure 1.

Let U_d be a network operating system utility to dismount a file and U_n a network operating system utility to mount a file.

The steps shown below must be followed to move a file from B_s to B_r.

1. The U_d routine is called by the B_s operator.
2. U_d initiates steps 1 through 6 and 8 of Procedure 1.
3. U_d notifies the operator of B_s that the file may be moved.
4. The B_s operator removes the file.
5. The B_r operator mounts the file.
6. The U_n utility routine is called by the B_r operator.
7. U_n initiates steps 10 through 12 of Procedure 1.

Constraints may exist on the movement of storage media among backend machines. Naturally, the physical limitation of device compatibility must be considered. A granule may be spread over multiple files, in which case all files must be moved together. This is determined from the list of file names maintained in the network data directory.

VI. Multiple Processor Backend

Throughout this paper, reference has been made to the backend machine rather than a backend processor. The reason for this distinction is that a backend machine may be composed of several processors. A multiple processor system may be configured to achieve one or several objectives; these include improved performance in parallel processing, reliability of excess resources (processes, memory, devices,

etc.) and locality of granules for fast access.

A multiple processor backend machine can be configured in several topologies--two of which are discussed here. First, a multiprocessor system where processors share all memory and devices provides complete recovery and parallel processing capabilities. This architecture is typically for homogeneous processors or a vendor generic family of processors. A second configuration of heterogeneous mini- and/or microprocessors can accommodate a multiple processor strategy via specialized memory-to-memory adapters (DMA) [29], or via loose-coupling connections (such as channels or telecommunications lines). The parallel processing strategy is more complex because each processor runs distinct software. As a backend, *one processor* must distribute requests for granule access across the affected processors and collect responses before returning responses to HINT. Recovery in this configuration will also require manual intervention (physical file movement) if one processor fails. Data movement among processors is not necessary either for performance or recovery reasons in the first configuration. Interprocessor movement of data could occur for either of the above reasons in the heterogeneous configuration. In such cases, data movement is accomplished by treating the multiple processor system as a small but complete distributed DBMS and applying Procedures 1 or 2.

Whereas performance considerations have not been stressed here, multiple processor backends must be configured with specific data distributions and a required performance in mind. The first system is only susceptible

to slow processor or slow device service due to request queues. The second architecture is susceptible to serious line delays because of loose couplings. Such an architecture also suffers from the overhead of interprocessor synchronization. However, this topology does not suffer device contention. Thus, tuning of such a system is via distribution of granules across processors, whereas the monolithic multiprocessor only distributes granules across devices.

In general, the performance of software functions in a multiple processor environment follows very much the same form as that described for the single backend environment. However, maintaining the support of a memory-to-memory communication link falls upon the network software. Further, other functions will cause more overhead as the level of complexity of the interactions increases.

VII. Data Integrity

A distributed data base system contains several potential hazards relative to the integrity of data, that either are not present or are less severe in a single machine DBMS. Some data transmission errors and the maintenance of redundant data items are unique to distributed systems. Although recovery and deadlock do require special attention in the single machine environment, their proper treatment in a distributed DBMS requires more complex measures. We now consider the ramifications of these hazards upon the procedures for data placement,

access, and movement.

The function of protecting the data base from the effects of data transmission errors lies with the communication system. The detection of major problems such as nonavailable links or grossly malfunctioning communication lines must be a basic part of any intermachine communication system. Failure to establish a link is indicated by a message from the communication system to the transmitting interface routine; this may either attempt a retransmission (after some delay) or abandon the communication. When a host interface detects a major communication failure, it returns an error code to the application program which permits the program to decide how to terminate gracefully. If the application program terminates due to communication failure, the HINT routine can initiate a rollback and recovery procedure when a communication path is available. A BINT routine does not initiate any recovery procedure upon detection of a communication failure. For reasons of data consistency, recovery must be initiated by the host computer [17]. The HINT routine awaiting the response from BINT could eventually detect the communications failure through a timeout mechanism and then initiate the recovery mechanism when communication is reestablished.

Multiple copies of specific data items in a network provide increased performance if the data items are accessed predominantly in a retrieval mode [3,5,15,21]. If a copy of the data item is placed on a backend physically proximate to the host machine which accessed the data, intermachine

communication would be reduced; however, update of redundant data poses some complex integrity problems. The mechanisms for operation in a distributed data base system presented in this paper do not consider the problem of data redundancy. In order to provide for the on-line update of redundant data, concurrency control methods similar to those presented in [1,5,7,25,26,28] would have to be employed.

The mechanism for data access presented in this paper does not consider the possibility of deadlock. In order to permit operation in a shared update environment, a concurrency control mechanism must be incorporated. The data movement procedures require the establishment of exclusive access to the granule prior to the transfer of data. Since the data movement procedure holds exclusive access to only one granule, no deadlock can occur due to data movement.

VIII. Conclusion

This paper has presented a mechanism for the distribution of a data base management system in a manner that is transparent to the application program. The software structures presented here presuppose an underlying computer network with the necessary hardware and software to allow interprocessor communication via a standardized message system. The basis for data base distribution is the network data directory which provides information on the location of each data base granule.

The mechanisms detailed here provide a data

distribution facility that is relatively easy to realize. However, many of the problems of distributed data bases [11,16,20,27] still require practical solutions. The dilemmas posed by deadlock, backup, recovery, and security are extremely complex. Another formidable stumbling block for distributed data base systems is the general lack of portability and compatibility within both hardware and software systems. The system described here is implementable on homogeneous networks with moderate effort. For heterogeneous networks, advances in software portability, hardware compatibility, and standardized communication protocols are required. Progress is being made in these areas although it is hampered somewhat by the marketing philosophy of "locking the user in" to a vendor's product line. Even within the product line of a single vendor, the appearance of commercial distributed data base systems is still some years away.

REFERENCES

1. Bernstein, P. A., et al., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The Fully Redundant Case)," IEEE Trans. on Software Engineering, (SE 4,3), May 1978, pp. 154-169.

2. Canaday, R. E., et al., "A Back-End Computer for Data Base Management," CACM, (17,10), Oct. 1974, pp. 575-582.

3. Casey, F. G., "Allocation of Copies of a File in an Information Network," Proc. AFIPS SJCC, Vol. 40, 1972, pp. 617-625.

4. Cerf, V. G. and Kahn, R. E., " Protocol for Packet Networks Intercommunication," IEEE Trans. on Communications (COM-22,5), May 1974, pp. 637-648.

5. Chu, W. W., "Performance of File Directory Systems for Data Bases in Star and Distributed Network," National Computer Conf., Vol. 45, June 1976, pp. 577-587.

6. Chu, W. W., and Ohlmacher, G., "Avoiding Deadlocks in Distributed Data Bases," ACM Annual Conf., Nov. 1974, pp. 156-160.

7. Ellis, C. A., "A Robust Algorithm for Updating Duplicate Data Bases," Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, pp. 146-158.

8. Fernandez, E. B., and Kasuga, H., "Data Control in a Distributed Database System," IBM L. A. Scientific Center G320-2693, Sept. 1977.

9. Hsiao, S. P., "Distributing a Data Base with Logical Assertions on a Computer Network for Parallel Searching," IEEE Trans. on Soft. Eng., June 1976.

10. Gray, J. N., Lorie, R. A. and Putzolu, G. R., "Granularity of Locks in a Shared Data Base, Conf. on Very Large Data Bases, Sept. 1975, pp. 428-451.

11. Hardgrave, W. T., "Distributed Database Technology: An Assessment," Information and Management, (1,4), Aug. 1978, pp. 157-167.

12. Karp. P. M., "Origin, Development, and Current Status of the ARPA Network," IEEE COMPCON, Mar. 1973, pp. 49-52.

13. Levin, K. D., and Morgan, H. L., "Optimizing Distributed Data Bases--A Framework for Research," National Computer Conf. (Vol. 44), June 1975, pp. 473-478.

14. Lowenthal, E. I., "The Backend Computer," MRI Systems Corp., P.O. Box 9968, Austin, TX 78766, Apr. 1976.

15. Mahmoud, S. A. and Riordan, J. S., "Optimal Allocation of Resources in Distributed Information Networks," ACM TODS, (11), Mar. 1976, pp. 66-78.

16. Maryanski, F. J., "A Survey of Recent Developments in Distributed Data Base Management Systems," IEEE Computer, (11,2), Feb. 1978, pp. 28-38.

17. Maryanski, F. J., and Fisher P. S., "Rollback and Recovery in Distributed Data Base Management Systems," Proc. ACM Annual Conf., Oct. 1977, pp. 33-38.

18. Maryanski, F. J.; Fisher, P. S.; and Wallentine, V. E., "Evaluation of a Conversion to a Back-End Data Base System," Proc. ACM Annual Conf., Oct. 1976, pp. 293-297.

19. Maryanski, F. J., et al., "Distributed Data Base Management Using Minicomputers," Infotech State of the Art Report Minis Versus Mainframes, 1978, pp. 141-157.

20. Miller, M. W., "A Survey of Distributed Data Base Management," Information and Management, (1,6), Dec. 1978, pp. 243-264.

21. Morgan, H. L., and Levin, K. D., "Optimal Program and Data Locations in Computer Networks," CACM (20,5), May 1977, pp. 315-322.

22. Peebles, R., and Manning, E., "System Architecture for Distributed Data Management," IEEE Computer (11,1), Jan. 1978, pp. 40-47.

23. Pouzin, L., "CIGALE, The Packet Switching Machine of the CYCLADES Computer Network," Information Processing-74, North-Holland, Aug. 1974, pp. 155-159.

24. Ries, D. P. and Stonebraker, M., "Effects of Locking Granularity in Database Management Systems," ACM TODS (2,3), Sept. 1977, pp. 233-246.

25. Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M., "A System Level Concurrency Control for Distributed Data Base Systems," Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, pp. 132-145.

26. Rosenkrantz, D. J., Stearns, R. E., and Lewis, P. M., "System Level Concurrency Control for Distributed Database Systems," ACM TODS Vol. (3,2), June 1978, pp. 178-198.

27. Rothnie, J. B. and Goodman, N., "A Survey of Research and Development in Distributed Data Base Management," Proc. Conf. on Very Large Data Bases, Oct. 1977, pp. 48-62.

28. Thomas, R. H., "A Solution to the Update Problem for Multiple Copy Databases," IEEE COMPCON, Mar. 1978.

29. Wallentine, V. E., et al., "An Overview of MIMICS," TR CS 77-04, Computer Science Department, Kansas State University, Manhattan, KS 66506, Mar. 1977.