DISTRIBUTED DATABASE MANAGEMENT SYSTEMS
ISSUES AND APPROACHES

Amjad Umar

The University of Michigan
Ann Arbor, Michigan 48109

Technical Report No. 88-8
July 1988

TABLE OF CONTENTS

DISTRIBUTED DATABASE MANAGEMENT SYSTEMS 2
1. INTRODUCTION 2
2. REVIEW OF BASIC CONCEPTS 3
2.1 DATABASE MANAGEMENT CONCEPTS 3
2.2 TRANSACTION MANAGEMENT CONCEPTS 5
3. DISTRIBUTED DATABASE CONCEPTS 6
3.1 INTRODUCTION 6
3.2 ADVANTAGES AND DISADVANTAGES OF DDBMS 6
3.3 REQUIREMENTS OF A GENERALIZED DDBMS 7
4. DDBMS ARCHITECTURAL CONSIDERATIONS 8
4.1 A REFERENCE ARCHITECTURE 8
4.2 USER INTERFACE MANAGER 9
4.2.1 Global Schema Definition 9
4.2.2 Global Directory Location 9
4.2.3 Discussion and Analysis 10
4.3. TRANSACTION PLAN GENERATOR 10
4.3.1 Methods for Accessing Remote Data 11
4.3.2 Remote Joins 11
4.3.3 Optimal Plan Generation 12
4.3.4 Discussion and Analysis 12
4.4 GLOBAL TRANSACTION EXECUTION MONITOR 13
4.4.1 Database Concurrency Control 13
4.4.2 Failure Detection and Transaction Recovery 17
5. DDBMS USAGE AND DISTRIBUTED DATABASE DESIGN 18
5.1 Data Allocation Strategies 18
5.2 Database Definition 19
5.3 Database Access and Manipulation 20
5.4 A Database Design Procedure 20
6. DDBMS IMPLEMENTATION CONSIDERATIONS 21
7 SUMMARY AND CONCLUSIONS 23

1. INTRODUCTION

Advances in the computer and communication technologies have led to distributed computer systems (DCS) which interconnect many small, medium and large computer systems through twisted pairs, cables, satellites, fiber optics and other transmission media. The major advantage of DCS is that one computer can share the resources (files, databases, programs, printers) of other computers. Resource sharing in DCS can be achieved at the following levels:

1. Terminal emulation where a computer C1 appears as a "dumb" terminal to another computer C2.

2. File transfer packages which allow a user to exchange a file F between computers C1 and C2. Users of file transfer packages must know the name of F and the identification of C1 and C2.

3. Servers (disk, file and print) which allow, usually on a local area network, several computers to simultaneously access files and printers located elsewhere. Servers usually allow access to "network" resources, say printer PTR5 and disk D, which are located at C1 but are available to C2.

4. Distributed Database Management Systems (DDBMS) which allow transparent access and manipulation of data that may be physically distributed to several local and, perhaps, heterogeneous databases.

Terminal emulation and file transfer have been available since the late 60s. Servers of several types are becoming commercially available with local area networks (see <Svobodova84> for extensive discussion of file servers). The main focus of this report is on DDBMS which are widely becoming commercially available.

DDBMS offer many advantages over the standalone databases, however several unique technical as well as management issues are raised. The literature on theoretical aspects, implementation issues, prototypes and commercially available DDBMS continues to grow steadily. However, it is difficult for a potential designer, manager or end-user of a DDBMS to understand and digest the technical body of knowledge about DDBMS due to formal content and differing terminologies.

The purpose of this report is to extract from the current literaure the key issues in developing and using DDBMS and identify the current approaches. This report is intended to serve three audiences. First, the developers of DDBMS who need to gain an indepth understanding of the subject area and who need to locate additional sources of information. Second, the managers who need to select a DDBMS package and who need to cut through the marketing literature in order to make sound decisions. Third, the programmers/designers of future application systems who will need to develop applications around DDBMS instead of the standalone or centralized databases. To accommodate different audiences, the treatment of the subject matter is informal. Numerous references are provided for additional information. The material is organized in the following sections:

- Section 2 introduces the basic database and transaction management concepts in centralized environments and defines the basic terms.

- Section 3 discusses the main DDBMS concepts and describes the goals of a generalized DDBMS.

- Section 4 describes the technical challenges faced by the designers and architects of DDBMS and reviews the approaches in developing DDBMS. Topics covered include directory definition, distributed query processing, concurrency control and failure handling. A DDBMS reference architecture is presented for systematic discussion of the subject matter.

- Section 5 concentrates on the issues related to the users of a DDBMS (data definition and distributed database design) and presents a general procedure for distributed database design.

- Section 6 reviews the facilities of the currently available DDBMS and section 7 concludes this report by summarizing the main points.

2. REVIEW OF BASIC CONCEPTS

Fig. 2.1 shows a typical environment with a database management system (DBMS) which manages the access and manipulation of a database and a transaction management system (TMS) which manages the interactions between the users and the DBMS. These two subsystems are described here to provide a basic conceptual framework which will need to be extended for DDBMS.

2.1 DATABASE MANAGEMENT CONCEPTS

Conceptually, a database consists of a collection of logical data items where the granularity of a logical data item may be a file, a record, or an arbitrary collection of data fields. A database management system (DBMS) is designed to:

1. maintain relationships between logical data items (one to one, one to many or many to many).

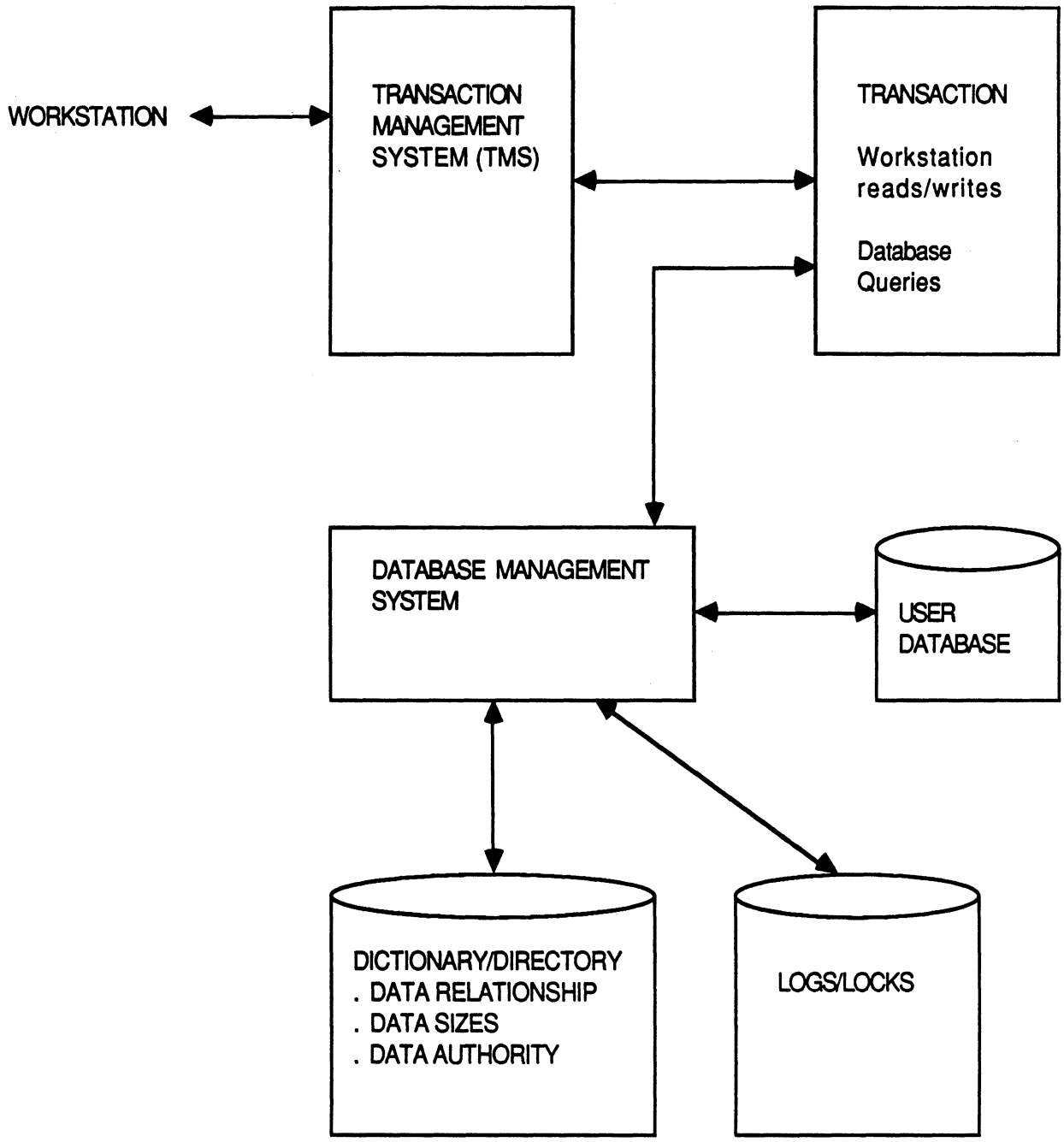
2. isolate programs from data format so that when data changes, the programs do not need to be changed

3. enforce security to allow access to authorized users only

4. provide integrity of the data in terms of data consistency (the data must correctly reflect the state of a system), currency (the database must contain recent information), and concurrency (the data must be simultaneously accessable by different users)

A typical database management system uses a database dictionary/directory to store the data relationships, data formats and security restrictions; database logs to record the activities of transactions; and lock tables to allow synchronous concurrent access to the database by several users (see Fig. 2.1). Facilities of a DBMS can be categorized in terms of data definition, data manipulation and operational facilities.

Fig. 2.1 Transaction Management system + Database Management



The Data Definition Language (DDL) is used to define the data formats and data relationships. Results of a DDL are stored in the dictionary. DDL allows data definitions at two levels: schema which shows the actual data format and data relationships and subschema which shows a user view of the data. Database management systems have traditionally supported the following three schemas:

- Hierarchical: the data is viewed so that only one to many relationships are supported. IBM's IMS <Kapp78> is an example.

- Network: Many to many relationships among logical data items are supported. Example of a network database management system is Cullinets's IDMS <IDMS81>.

- Relational: the data is viewed as tables (relations). Example of a relational system is IBM's DB2 <Date84>.

In addition, a great deal of work at present is being focussed on the development of "object oriented knowledge bases" which allow storage of objects, relationships between objects, rules and events for complex engineering, business and expert systems applications.

The Data Manipulation Language (DML) allows a user to access and manipulate the database. The power and capability of the DML depends on the views supported (hierarchical, network, relational). For example, the relational systems allow a user to access any information from the database with only three operations:

- selection which chooses rows of a table based on a criteria

- projection which chooses columns of a table based on a criteria

- join which combines two different tables on a common attribute

Additional operations like union and semi-join <Bernstein81b> may also be supported by some relational systems. For example, semi-join is introduced to minimize the internode traffic while performing a join of two remotely located tables. Semi-join will be explained in section 4.3.

The Operational Facilities of a DBMS provide security, concurrency and backup/recovery of a database. The operational facilities of a large centralized DBMS must be comprehensive enough to allow simultaneous access of hundreds of users to large centralized databases. On the other hand, operational facilities may not be needed for single user microcomputer databases.

Different commercial DBMS allow different levels of DDL, DML and operational support. Some general observations about the commercial DBMS are:

1. Relational systems are most popular and relational DML SQL is fast becoming the standard in relational systems. Information about SQL can be found in <Date84, Date86>.

2. Most microcomputer database systems are relational, are designed for single user, support interactive DML and DDL, do not support programming interface (DML) and are weak in operational support like security and backup and recovery.

3. Most host DBMS are becoming relational, are designed for multiple users, support program imbedded DML and have strong operational features.

Despite the commercial popularity of relational systems, the weaknesses of relational systems for CAD/CAM applications are being recognized. For

example, CAD/CAM applications require very long records (several megabytes), CAD/CAM records are of variable length, CAD/CAM relationships are complex and version control/archiving of portions of CAD/CAM designs are important <Wolfe87>. Several companies (IBM, Computervision, Sherpa Corporation, and Digital Equipment Corporation are developing CAD/CAM database products on top of existing commercial DDBMS. In addition, relational systems are not adequate for knowledge engineering applications which require semantic information and inference.

The database concepts in this section have been introduced informally. The interested reader is referred to several books <Kroenke77, Vasta85, Date84, Date86, Ullman83, Martin86, Teorey82> for additional information.

2.2 TRANSACTION MANAGEMENT CONCEPTS

The concept of a transaction originates from the field of contract law <Gray81, Walpole87> where each contract between two parties (a transaction) is carried out unless either party is willing to break the law. These transactions have three properties: consistency (they must obey the law), atomicity (they either happen in entirety or do not happen at all), and durability (once committed they cannot be cancelled). In computer science, a transaction is defined as a sequence of database queries (data access and manipulation commands) that transforms one consistent state of the system into a new consistent state <Eswaran76>. These transactions must also observe the consistency, atomicity and durability restrictions.

Transactions may be on-line queries or application programs written in a procedural language with imbedded queries. A transaction management system (TMS) provides the following facilities:

1. Schedule transactions for execution. Transactions can be scheduled serially ("single-threaded") or in parallel ("multi-threaded").

2. Allow read/write operations from the user workstations. Each workstation may be a terminal or a computer. Workstation read/write operations may involve the communication activities of establishing sessions, determining network routes and assembling data link control messages. These operations can be described by using the layers 5 through 1 of the ISO/OSI Distributed Reference Model <Tannenbaum81>.

3. Manage the resources needed by the transactions during execution.

These TMS facilities are commonly integrated with the DBMS facilities, as shown in Fig. 2.1, which allow database queries from the transactions. These queries may involve one or several data items. The transaction may be decomposed into subtransactions to optimize I/O and/or response time (see <Jarke84> for comprehensive discussion).

An example of commercially available TMS is IBM's CICS <Lim82>. It is not always possible to find separate TMSs in commercial products. In several cases, TMS facilities are imbedded in communication managers, operating systems and/or database managers. An introduction to TMS facilities is given by <Stamper86>. For an extensive discussion of transaction management, see <Gray81, Walpole87>.

3. DISTRIBUTED DATABASE CONCEPTS

3.1 INTRODUCTION

A Distributed Database Management System (DDBMS) allows a user to access and manipulate data from several databases that are physically distributed to several sites. A DDBMS differs from a DBMS, henceforth referred to as a local DBMS (LDBMS), because it is responsible for providing transparent and simultaneous access to several databases that are located on, perhaps, dissimilar computer systems where a local DBMS manages a single site database. An example of a DDBMS is illustrated in Fig. 3.1 where a parts database is located on a network of three computers - an IBM mainframe under MVS, a VAX system under VMS and an IBM PC under DOS. The three computers may use three different LDBMSs: DB2, Ingres and Dbase3. This DDBMS would allow an end-user to:

1. create and store a new part anywhere in the network

2. access a part-no without knowing where the part-no is physically located

3. delete a part from one database without having to worry about the duplicated parts in other databases

4. update a part description in one database without having to worry about how and when all the duplicated parts will be updated in other databases

5. access a part from an alternate computer when, say, the nearest computer is not available.

In the future, a common problem for DDBMS will be to provide transparent access to the commercial, engineering and expert system databases that may be stored anywhere in the network.

3.2 ADVANTAGES AND DISADVANTAGES OF DDBMS

It can be seen that the DDBMS environment offers several advantages over the LDBMS. For example, a DDBMS:

1. saves communication costs by providing data at the sites where it is most frequently accessed

2. improves the reliability and availability of a system by providing alternate sites from where the information can be accessed

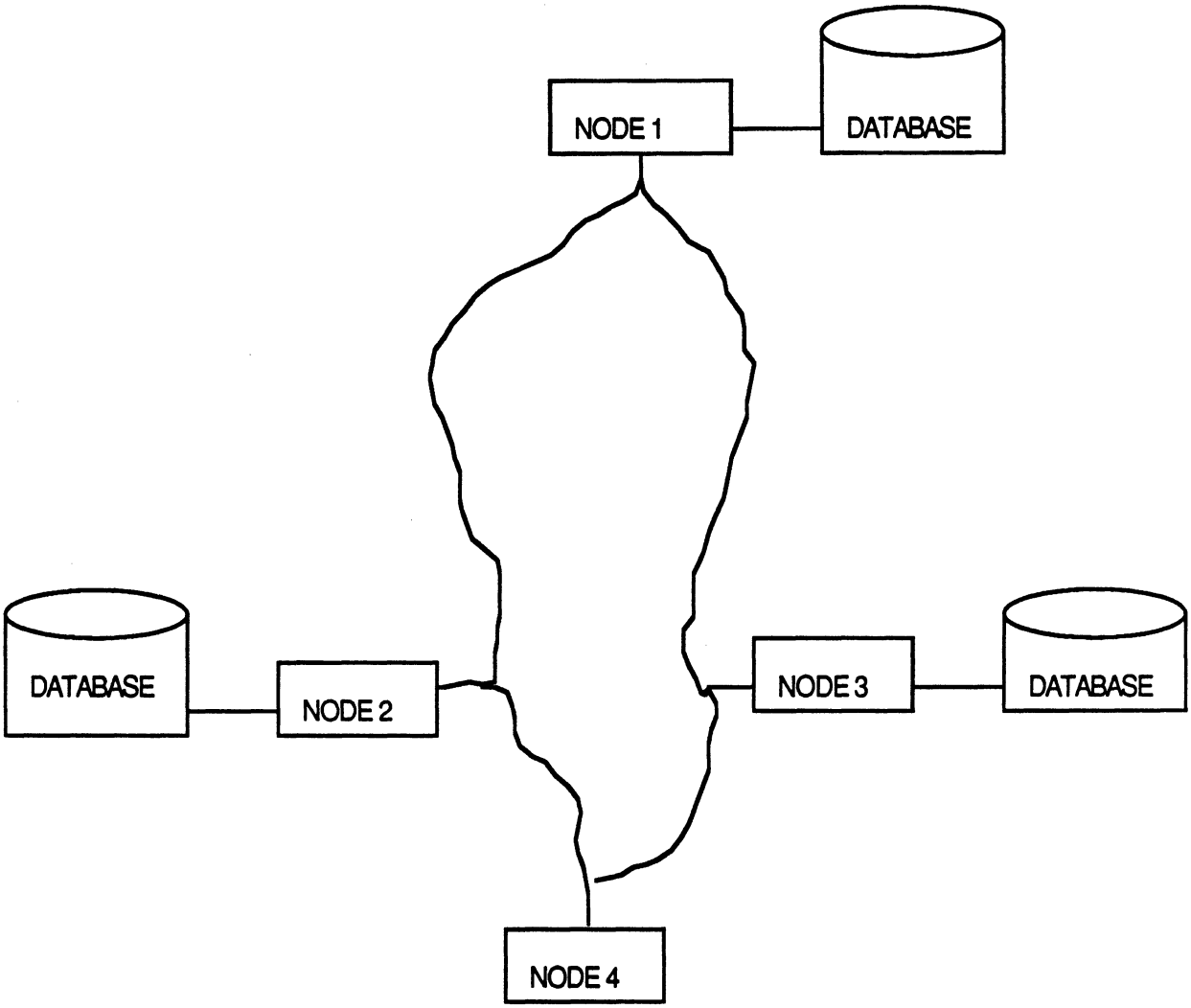
3. increases the capacity of a system by increasing the number of sites where the data can be located

4. improves the performance of a system by allowing local access to frequently used data

5. allows users to excercise control over their own data while allowing others to share some of the data from other sites

However, a DDBMS has the follwing disadvantages:

Fig. 3.1: An Example DDBMS



1. increases the complexity of the system and introduces several technical as well as management challenges especially when geographical and organizational boundaries are crossed

2. makes central control more difficult and raises several security issues because a data item stored at a remote site can be always accessed by the users at the remote site

3. makes performance evaluation difficult because a process running at one node may impact the entire network

4. may deteriorate the overall performance of wide area networks (WANs) which suffer from the "SUE" factor (slow, unreliable, expensive) as pointed out by Gray <Gray87>.

It should be emphasized that due to the advantages/disadvantages of a DDBMS each application of DDBMS must be carefully evaluated.

3.3 REQUIREMENTS OF A GENERALIZED DDBMS

The following functional and operational requirements of a generalized DDBMS are stated to provide a framework for discussion:

Functional Requirements:

1. The DDBMS should be able to support heterogeneous underlying LDBMS

2. The user should be able to access data transparently without knowing the actual location and internal format

3. The user should be able to allocate single copy, two copies (one at local node, another at central), or any number of copies in the network

4. The user should be able to partition and fragment tables in the network

5. A common global user view of the data should be supported

6. A common global data manipulation language should be supported

7. The user should be able to issue distributed transactions which require information from several nodes

Operational Requirements:

1. Many users should be able to simultaneously update the duplicated database.

2. The data integrity should be supported at several levels: transaction level (all duplicate updates must be applied before this transaction ends) and hourly/daily.

3. The distributed transaction processing should be optimized to minimize query processing costs (time, communication I/O, etc)

4. The communication and node failures should not drastically impact the the operation of the DDBMS

5. The DDBMS should use standard communication protocols and should be transportable among different systems

6. The software design of DDBMS should maximize use of asynchronous and concurrent processing

It should be noted that these requirements are listed for conceptual understanding of DDBMS. A given DDBMS may choose to implement only a small portion of these requirements depending on the potential application of DDBMS.

4. DDBMS ARCHITECTURAL CONSIDERATIONS

An architecture of DDBMS can be developed to satisfy the requirements stated in section 3.3. This architecture will partition the requirements stated above to various components. Several architectures of DDBMS have been presented <Bernstein81, Decitrye83, Gligor83, Larson85, Cardenas87>. A reference architecture is presented in section 4.1 to describe the main components of a DDBMS. This reference architecture is an extension of the architecture presented by Gligor and Fong <Gligor83>.

4.1 A REFERENCE ARCHITECTURE

Fig. 4.1 shows a reference architecture which identifies the main components of a DDBMS and shows how these components interact with the Local Transaction Management system (LTMS) and Local Database Management System (LDBMS). This architecture is presented as sublayers of layer 7 (application layer) of the ISO/OSI Distributed Reference Model. A Remote Access Interface is introduced to provide the interface with remote sites. A brief overview of the architecture is presented here and details are given in sections 4.2 through 4.4.

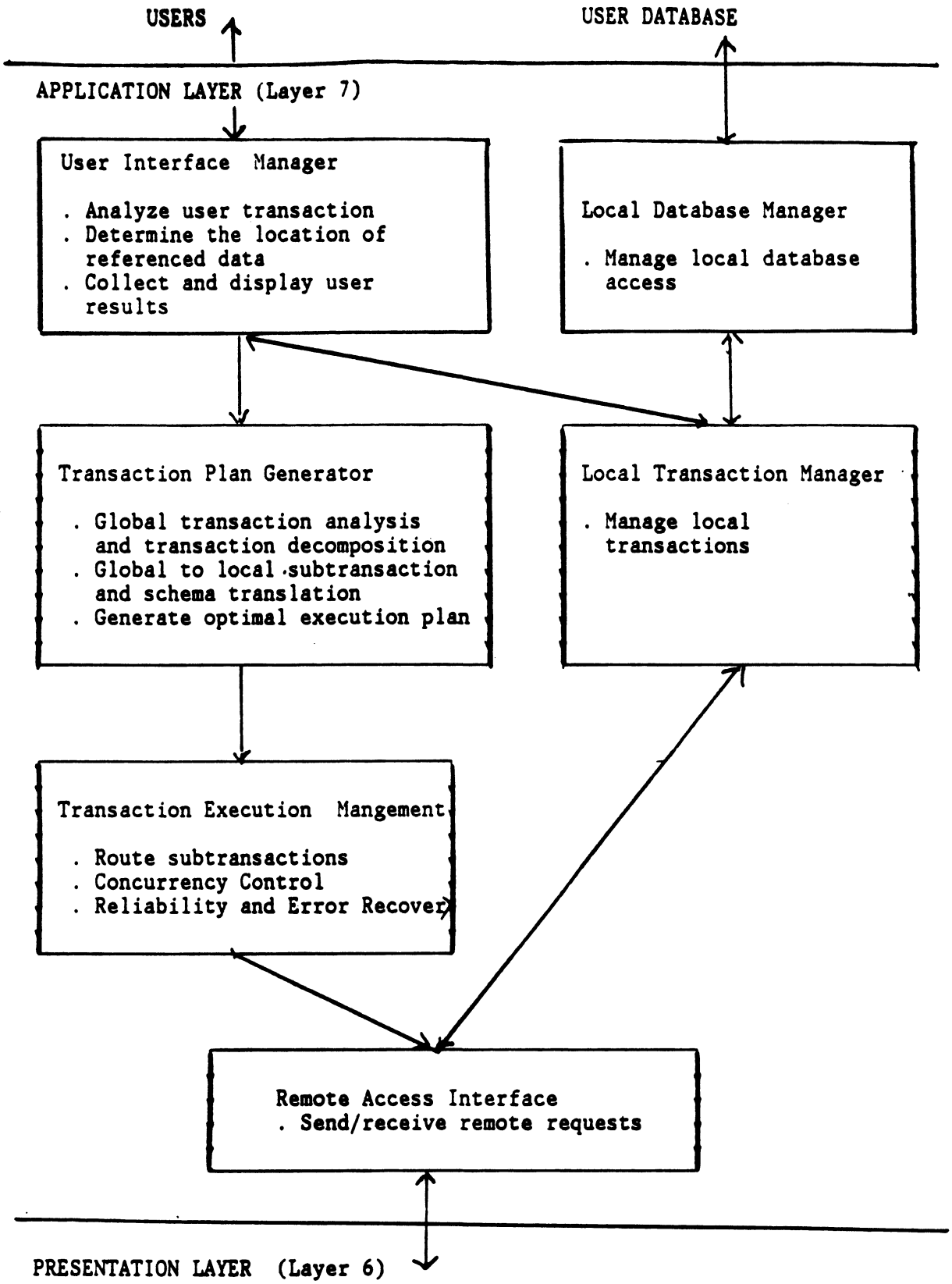
User Interface Manager (UIM). This "module" translates the queries into a global form if necessary, determines the location of the data referenced in the queries, passes control to LTMS if the transaction is local only or to the Global Transaction Analyzer (GTA) if the transaction needs access to remotely located data. This module is also responsible for gathering all user results generated during transaction execution and presenting the results to the user.

Transaction Plan Generator (TPG). This module is responsible for generating an execution graph (plan) to optimize the performance of the arriving transaction. This may involve decomposing the transaction into subtransactions which can run as local transactions on various nodes and translation of the global transaction into local transactions.

Global Transaction Execution Monitor (GTEM). This module receives the plan generated by TPG and is responsible for the initiation, execution and integrity control (synchronization, reliability) of the transaction plan.

Remote Access Interface. This module prepares all of the messages that need to to be sent to remote nodes in a particular format and also receives the messages from the presentation layer and passes them to LTMS or GTEM sublayers. The protocols used in this sublayer may be the File Transfer Access Method (FTAM) of ISO, the Manufacturing Message System

Fig. 4.I : A Reference Architecture



of Manufacturing Automation Protocol (MAP), or the X.400 protocol proposed for electronic mail.

The Local Transaction Management System (LTMS) module performs the functions described in section 2.2 and the Local Database Management System (LDBMS) performs the functions described in section 2.1.

4.2 USER INTERFACE MANAGER

This module performs the following functions:

1. Read and parse the user transaction.

2. Determine the location of the data referenced by this transaction. If all referenced data is local, then pass control to LTMS, else pass control to Transaction Plan Generator sublayer.

3. Collect and present results to the user.

The two major issues are the global schema definition and allocation.

4.2.1 GLOBAL SCHEMA DEFINITION

A common global schema is needed to parse the query in global query format. The global schema shows all the data in the network and shows where the data is located. The problem of global schema design is straightforward if all the LDBMS are homogeneous but is nontrivial in a network with heterogeneous databases. At present most DDBMS are using SQL for schema definition and manipulation <Gligor83, Rauch-Hinden87>. The interested reader is referred to <Cardenas87> for schema design considerations for heterogeneous DDBMS.

4.2.2 GLOBAL DIRECTORY LOCATION

It is customary to show the (data, node assigned) pair in the global schema and store the global schema in a global directory. Due to the number of global directory accesses, it is crucial to allocate the global directory carefully. The directory allocation problem can be treated as a file allocation problem (FAP) where a file F is allocated to N nodes to minimize a given objective function. For example, Chu <Chu76, Chu84> has studied the directory allocation problem as a FAP. The following tradeoffs can be observed:

1. If the directory is at a central site, then the communication cost is high because every transaction will need to access the central site to locate data.

2. If the directory is at every site, then the update cost will increase due to duplicate directory updates.

It is common to replicate the global directory even if the data is not replicated. Another common approach is to store the directory at a central node with the following processing rules:

. search the local directory at the arriving node
. if not found then search the directory at the central site

It has been found that the directories can afford to be inaccurate for a while because they are treated as cache <Goodman87>.

4.2.3 DISCUSSION AND ANALYSIS

The design of this module is dependent on the DDBMS requirements of supporting heterogeneous LDBMS and common global views. This issue is important at the transaction parsing and result generation steps.

The user transactions may appear in a uniform global query format, e.g. SQL, or in format of the LDBMS, e.g. Dbase3 format on an IBM PC and IMS format on IBM mainframe. The uniform global query format is preferable even though the user of an existing LDBMS may need to learn a new query language.

The results of a transaction are collected and presented to the user at the termination of the transaction. A considerable result translation may be needed if the transaction accessed data, located in several heterogeneous databases.

4.3. TRANSACTION PLAN GENERATOR

The transaction plan generator is a crucial component of DDBMS and performs the following functions:

1. Determine the method of accessing data from remote sites.

2. Decompose the transaction into subtransactions if needed.

3. Translate the global transactions into local transactions if needed.

4. Generate an execution plan to optimize response time and/or communications cost.

To illustrate the processing of this sublayer, assume that a transaction T arrives at node n0 and accesses logical data items d1 and d2 where d1 is located on node n1 and d2 is located on node n2. T can be processed by using one of the following plans:

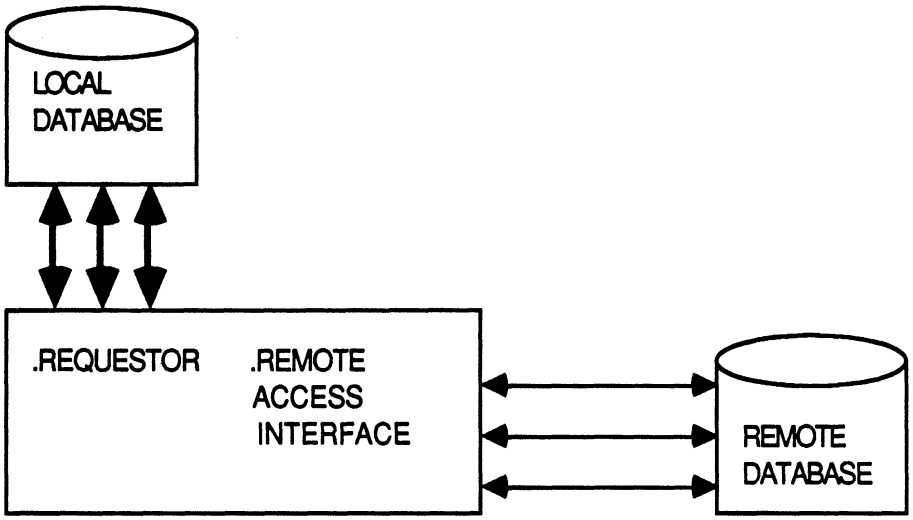
1. T is executed at n0 and issues remote calls to n1 and n2

2. T is routed to n1 where it accesses d1 and issues remote calls to node n2

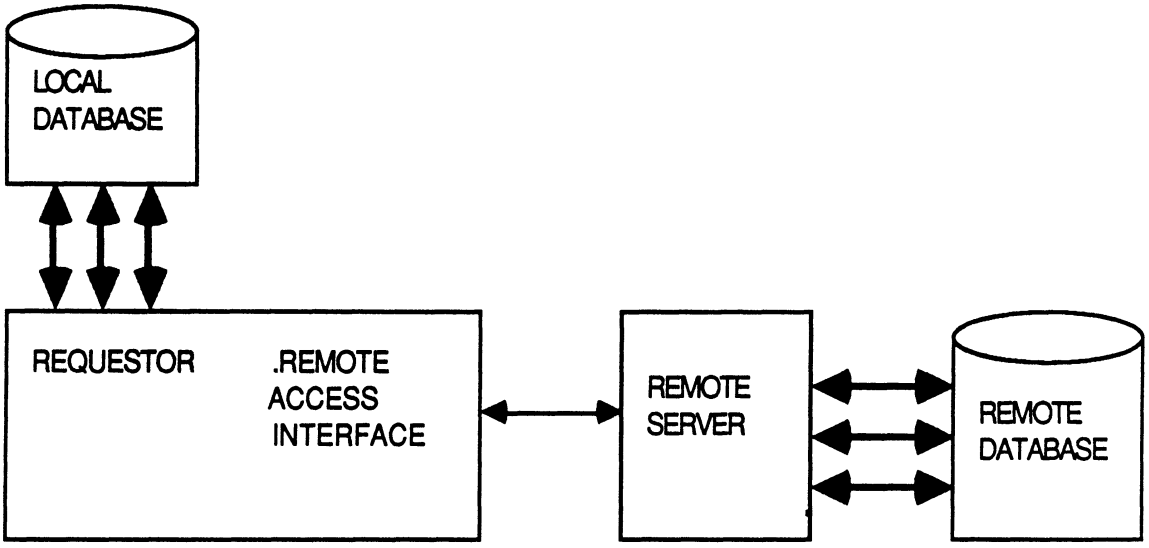
3. T is routed to n2 where it accesses d2 and issues remote calls to node n1

4.1: REMOTE ACCESS METHODS

a). Transparent Access



b). Remote Procedure Call (Requestor/Server)



Local i/o

Remote i/o

4. T is decomposed into t1 and t2 where t1 accesses d1 and t2 accesses d2; t1 is sent to n1 and t2 is sent to n2.

The Transaction Plan Generator will determine and generate one of these transaction processing plans based on an optimality criteria. In general, the problem of determining optimal transaction processing plans, known in literature as "distributed query processing" <Yu84, Hevner81>, is reverse of the optimal data allocation problem because in data allocation the data is allocated to nodes based on known queries while in transaction processing the optimal plan is determined based on known data allocations. This is why some combined data allocation and query optimization algorithms have been proposed <Hevner81>.

The transaction plan generation is dependent on three major issues: methods used to access remote data, the methods used for performing remote joins and the optimization strategy.

4.3.1 METHODS FOR ACCESSING REMOTE DATA

The three basic methods for accessing remote data are transparent remote access, remote procedure call and transaction routing (see Fig. 4.1).

Transparent Remote Access. This method provides a "location-independent" transparent access to data located anywhere in the network. As shown in Fig.4.1a, every data access request (e.g. SQL Select) is sent to the remote node through the Remote Access Interface of the Reference Architecture. This method is very convenient but may cause excessive communication traffic.

Remote Procedure Call.This method allows the data at a remote site to be accessed locally by a remote server as shown in Fig. 4.1b. The requester of a remote procedure call, also called a "client", is responsible for preparing and sending the remote procedure call. It can be seen from Fig. 4.1b that a remote procedure call can minimize the communication traffic. An extensive discussion of remote procedure calls can be found in <Wilbur87>.

Transaction Routing. By using this method, a transaction is decomposed into subtransactions where each subtransaction accesses data at only one node. Then each subtransaction is routed to the remote site and the results are sent back to the destination node. This approach maximizes the concurrent operations of transactions. Decomposition strategies have been discussed extensively in the literature (see <Yu84, Sacco81, Hevner85, Chu84>).

4.3.2 REMOTE JOINS

Since joins between remote sites can cause a considerable amount of communication traffic, a semi-join has been proposed to send only the necessary data <Bernstein79>. Fig. 4.2 illustrates the semi-join operation. Simply stated, it consists of the following steps for joining R1 and R2 on attribute A:

1. Relation R1 is projected on attribute A, giving R1'

2. R1' is transmitted to R2 and joined with R2, giving R2'

Fig. 4.2: Semi-Join Operation

Join Relation PAYROLL at node n1 with PERSONEEL at node n2

PERSONEEL (n1)

NAME	EDUC.	ADDRESS
Sam	B.S.	Detroit
Joe	B.S.	NY
Pat	MBA	Toledo
Bob	B.S.	Detroit
Jack	M.S.	LA

PAYROLL (n2)

NAME	POSITION	SALARY
Joe	Programmer	30k
Pat	Manager	40k

STEPS IN SEMI-JOIN:

1). Project PAYROLL on NAME at n2: gives PAYROLL' = NAME

PAYROLL'

Joe
Pat

2). Transmit PAYROLL' to n1

3). Join PAYREOLL' with PERSONEEL at n1, gives PERSONEEL'

PERSONEEL'

NAME	EDUC.	ADDRESS
Joe	B.S.	NY
Pat	MBA	Toledo

4). Send PERSONEEL' to n2

5). Perform a join between PERSONEEL' and PAYROLL at n2
Gives the final ansewr

NAME	EDUC.	ADDRESS	POSITION	SALARY
Joe	B.S.	NY	Programmer	30k
Pat	MBA	Toledo	Manager	40k

3. R2' is transmitted to R1 and joined with R1, giving the final join.

Translation of a join into semi-join is an example of query decomposition.

4.3.3 OPTIMAL PLAN GENERATION

Determination of an optimal transaction processing plan is a difficult problem. Examples of existing methods are:

1. The "hill-climbing" algorithm developed for SDD-1. This algorithm uses semi-joins to reduce the data transmitted and chooses a final node where all the relations are transmitted for final joins <Goodman79, Wong77>.

2. A family of algorithms to minimize either response time or total time for "simple queries" in which each relation contains only one common joining attribute, which is also the output of the query <Hevner79, Yao79>.

3. The evaluation algorithm used in Distributed Ingres <Epstein80>. This algorithm fragments one relation and replicates others so that each site containing a fragment and the replicated relations can process the join in parallel.

These algorithms differ in several assumptions: network topology, result site, cost function, and objective function. Detailed analysis of these algorithms can be found in <Sacco81>. Recent trends in query optimization indicate heavy usage of artificial intelligence techniques like statistical information about data distribution, explanation and inference <Epstein87>.

4.3.4 DISCUSSION AND ANALYSIS

It is difficult to make general remarks about query optimization algorithms because each algorithm must be evaluated for given application and network type. For example, Stone <Stone87> recently describes a case study in which an efficient serial algorithm performs better than a parallel query processing algorithm running on 64,000 processors. It is common to devise transaction processing networks which minimize communication cost and maximize use of CPU for slow networks; and minimize response time for fast networks.

It appears that duplicating data unnecessarily increases the complexity of query optimization and adds to the overhead. Although academic literature concentrates heavily on query decomposition and optimization techniques, several heavily used industrial systems like Tandem Encompass and IBM CICS/ISC heavily use remote procedure calls where data is managed by autonomous remote servers with minimum internode communications. IBM's SNA LU6.2 feature, also known as Advanced Program to Program Communication (APPC), is essentially a remote procedure call mechanism which is being adopted by major vendors as the server-requester protocol. IBM has also announced an array of products based on LU6.2: distributed database managers (DDM), transaction processors (CICS), electronic mail systems (DISOS), and document interchange mechanisms (DIA/DCA). A major supporter of remote procedure calls is Jim Gray <Gray87> who claims that unnecessary data duplication is too expensive and the benefits in response

time and availability are not worth the added overhead. This needs to be weighed in terms of the underlying network: WAN or LAN.

To summarize, transaction plan generation can be extremely complex if data is duplicated and dispersed at several sites. For most practical purposes, it may be best to assign data uniquely where it is most frequently used and utilize an efficient remote procedure call facility. The interested reader is referred to growing literature in this area for additional information <Hevner87, Yu84, Ceri84, Sacco81, Hevner79>.

4.4 GLOBAL TRANSACTION EXECUTION MONITOR

This module receives the plan generated by the Transaction Plan Generator and is responsible for monitoring the execution of the plan. It performs the following functions:

1. The transactions or subtransactions are initiated in the manner indicated by the plan.

2. The concurrency control and deadlock detection/resolution of the executing transactions are managed.

3. All failure and/or performance problems are monitored and handled.

4. All results collected from the transaction execution are sent to the User Interface Manager.

This module closely interacts with Remote Access Interface and LTMS. The two main issues addressed by DTMS are the database concurrency control and reliability and error recovery. These two issues are discussed in sections 4.4.1 and 4.4.2, respectively.

4.4.1 DATABASE CONCURRENCY CONTROL

Concurrency control coordinates simultaneous access to shared data. At present, the problem of concurrency control in centralized DBMSs is well understood and one approach, called two-phase locking has been accepted as a standard solution <Eswaran76>. However, concurrency control in distributed systems is an area of considerable activity with no accepted solutions. This is due to three main complicating factors. First, data may be duplicated in DDBMS, consequently, the DDBMS is responsible for locating and updating the duplicate data. Second, if some sites fail or if some communication links fail while an update is being executed, the DDBMS must make sure that the effects will be reflected on the failing node after recovery. Third, synchronization of transactions on multiple sites is very difficult because each site cannot obtain immediate information on the actions currently being carried out on other sites.

Due to these difficulties, over 20 concurrency control algorithms have been proposed in the past, and still continue to appear (see, for example, <Farrag87>). Literature surveys have shown that most algorithms are variants of two-phase locking and time stamped algorithms <Bernstein81, Hsiao81>. However, several algorithms do not fall into any category. The three categories are briefly described and analyzed below. For a very detailed discussion of concurrency control algorithms, refer to <Bernstein81, Kohler81, Hsiao81, Bernstein87>.

4.4.1.1 Two-Phase Locking Mechanisms

Two-phase locking (2PL) uses read and write locks to prevent conflicts between concurrent operations. The 2PL algorithm consists of the following steps for a transaction T in an LDBMS:

. T obtains a readlock to data item x
. T obtains a writelock to data item y
. T reads x
. T writes y
. T releases the readlock on x
. T releases the writelock on y

The lock ownership is governed by two rules: (1) different transcations cannot simultaneously own conflicting locks (read-write or write-write conflicting locks); and (2) once a transaction surrenders a lock, it may never obtain additional locks. This implies that each transaction goes through two distinct phases: a "growing" phase when a transaction obtains locks and a "shrinking" phase when the transaction releases locks.

The 2PL algorithm can be implemented in DDBMS by using the following scheme:

. T obtains a readlock for x at node n where it is going to be read
. T obtains a writelock to all copies of data item y
. T reads x
. T writes y
. T updates all duplicate copies of y
. T releases the readlock on x
. T releases the writelock on all copies of y

Several variants of basic 2PL have been proposed:

1. The Primary Copy 2PL - a primary copy is designated for each data item x; read locks are granted by the primary node and the update locks are granted at each site where the data (primary or duplicated) is stored. This algorithm has been proposed by Stonebraker <Stonebraker79>.

2. The Voting 2PL - read/write lock requests are sent to all nodes; if majority returns with lock granted, then lock is granted else transaction waits for locks. This algorithm is also referred to as the Thomas Voting algorithm <Thomas79>.

3. Centralized 2PL - the lock manager and lock tables are kept at a central node; if a transaction needs to access any x in the network, it must be granted lock from the central site. An extensive study of centralized 2PL locks have been conducted by Garcia-Molina <Garcia79> who found that the performance of centralized locking schemes is better than expected.

Several issues arise in the implementation of 2PL-based concurrency control algorithms:

1. What is the locking granularity, i.e. the smallest database object that can be locked (database, file, record, field)?

2. What is the update synchronization interval, i.e. the time elapsed before updates on x are applied to all copies of x. The updates may

need to be applied at transaction run, hourly or even daily, depending on the type of application.

3. Where should the lock tables be located?

4. Will a deadlock be caused due to locking? Simply stated, a deadlock occurs in the following situation:

   . T1 must wait for T2 to release a lock
   . T2 must wait for T3 to release a lock
   . T3 must wait for T1 to release a lock

   Deadlocks may occur more frequently in 2PL based algorithms because all transactions obtain all locks in the growing phase thus several transactions may wait on each other for locks. Deadlock prevention and detection techniques are used frequently in DDBMS. No accepted solution exists for this problem. One of the simplest techniques is to use "timeout" which forces a transaction to abort after it has waited for more than a timeout limit.

Detailed mathematical models and simulation results have been developed to determine which algorithm is best for what type of application with different granularities, synchronization intervals and lock table locations <Garcia79>. Recently, Wolfson <Wolfson87> has given an extensive analysis of locking and commit protocols in DDBMS. Although it is difficult to make general observations, most of the available DDBMS use locking mechanisms which are variants of 2PL <Goodman87>.

4.4.1.2 Time Stamped Algorithms

Each transaction T is assigned a unique timestamp at the originating node. This time stamp is attached to each read and write request. In case of a conflict, i.e. two transactions trying to update the same data item x, the conflicting requests are serialized by time stamp. Although several time stamped concurrency control algorithms have been proposed <Bernstein81, Hsiao81>, only a few have been actually implemented in commercial DDBMS. The major reason for this may be that time stamped algorithms require a universal clock which is generally not available on most existing distributed systems.

4.4.1.3 Miscelaneous Concurrency Control Algorithms

Several algorithms have been proposed which combine the time stamped and 2PL algorithms, use unique approaches like Ellis Ring Algorithm <Ellis77> or use certain pessimistic/optimistic assumptions about the nature of the sytem. An interesting example of a generalized concurrency control algorithm has appeared recently <Farraag87>. This algorithm considers the 2PL and time-stamped algorithms as special cases. Due to space limitations, only pessimistic/optimistic will be discussed here. The interested reader should refer to <Bernstein81, Hsia81> for others.

The pessimistic algorithm requires that a transaction secure all the nececessary locks on data items before execution. If all the locks cannot be acquired then the transaction will not start execution. This algorithm can be represented by noting that a "lock" transaction T' is started before T just to obtain locks:

. T' obtains a readlock to all data items x needed by T
. T' obtains writelock to all data items y needed by T
. T starts execution
. T reads x
. T writes y
. T writes all duplicate copies of y
. T releases the readlocks on x
. T releases the writelocks on y

This algorithm is called pessimistic because it assumes that a large number of conflicts will be encountered by T, thus T should not be started without owning all locks.

An optimistic algorithm assumes that very few conflicts will occur and operates in the following manner.

. T starts execution
. T reads x at initiating site
. T writes y at initiating site
. The new values of y are sent to all nodes with duplicate y
  The new values are sent with a time stamp
. If the new values are updated correctly with no conflicts,
  then the updates are made permanent
  otherwise the update is discarded and applied again

It can be seen that the optimistic algorithm will work well if there are very few conflicts. However, in case of several conflicts, there is no guarentee that a transaction will complete in finite time due to repeated restarts <Milenkovic80>.

4.4.1.4 Discussion and Analysis

As mentioned previously, many database concurrency control algorithms have been published and continue to appear in computing journals. It is difficult to evaluate these algorithms because they use different terminologies, make different assumptions about the underlying DDBMS, are hard to understand and are difficult to prove correct. Each author claims that his/her approach is the best. In addition, it is difficult to determine which of these algorithms has been implemented and which implementations are being used in commercial products instead of laboratory prototypes.

For a prospective DDBMS developer, the following is a suggested approach:

1. Determine the real need for data duplication. The cost of synchronizing duplicated data in several cases outweighs the benefits.

2. Carefully examine the synchronization interval. Most concurrency control algorithms assume that data consistency needs to be maintained within a transaction. For data with long synchronization intervals, no concurrency control is needed.

3. For the DDBMS needing concurrency control, use a variant of 2PL, perhaps the primary copy 2PL because it is used most frequently.

4. The optimistic concurrency control algorithms seem to work very well when the transactions are short and the network is fast and reliable.

5. If possible, develop a simple analytical or simulation model to measure the effect of concurrency control algorithms. In most cases,

the analytical models are quite complex <Umar84, Garcia79>. It may be better to develop a simulation model.

4.4.2 FAILURE DETECTION AND TRANSACTION RECOVERY

In an LDBMS, updates are made permanent when a transaction commits, and updates are rolled out if a transaction aborts. In DDBMS, a transaction may commit at one node and abort at another. For example, update completes at node n1 and fails at n2.

A transaction may terminate abnormally due to two reasons: "suicide" which indicates that a transaction terminates due to an internal error like a program check, or "murder" which indicates an external error like system crash <Gray79>. It is the responsibility of DDBMS to remove all changes made by a failing transaction from all nodes so that the transaction can be reinitiated.

In order for atomic actions to be recoverable, the following two conditions must be met:

1. Updated objects are not realeased until the action is completed

2. The initial states of all objects modified by the action can be reconstructed through the use of a log.

An extensive recovery system for distributed database management systems has been proposed by Gray <Gray79> and implemented in System R. This system consists of the following four protocols:

1. Consistency Locks: This means that each transaction must be well formed and two-phase. A transaction is well formed if it locks an object before accessing it, does not lock an already locked item, and unlocks each locked item before termination. A transaction exhibits two-phase behavior if no objects are unlocked before all objects are locked. Transactions using 2PL are automatically well formed and two-phase.

2. DO-UNDO-REDO log: This is an incremental log of changes to the databases which records the before/after images of each update during the transaction processing (DO operation). This log also allows removal (UNDO) of a failed transaction's updates and reapplication (REDO) of the successful transaction's updates.

3. Write-Ahead Log: This protocol consists of writing an update to a log before applying it to the database.

4. Two-Phase Commit: This protocol coordinates the commit actions needed to run a transaction. When the transaction issues a COMMIT request, then the following actions take place:

```
Phase 1:
   1). Sends a PREPARE message to each cohort (same process
       running on another node)
   2). Wait for a reply from the cohort
Phase 2:
   1). Write COMMIT entry into the log
   2). Send a COMMIT message to each cohort
   3). Wait for positive response from each cohort
   4). Write a complete entry in log and terminate
```

The two-phase commit has been implemented in several other systems in addition to system R. Other approaches have been proposed and are being investigated. An extensive discussion of the reliability of distributed database management systems for no data replication, data replication, full replication, and network partitioning is given by Garcia-Molina <Garcia87>.

5. DDBMS USAGE AND DISTRIBUTED DATABASE DESIGN

Section 4 concentrated on the architectural and design aspects of DDBMS. Once a DDBMS has been developed or purchased, then the users (application designers, database designers, administrators, application users) face three major decisions: where to allocate the database, how to define the different views of the database and how to access and manipulate the database. These decisions are significantly influenced by the architecture and the facilities of the DDBMS and are discussed in sections 5.1 through 5.3. A simple distributed database design procedure is given in section 5.4.

5.1 DATA ALLOCATION STRATEGIES

It was suggested in the previous section that the data allocation decision significantly impacts the query processing, concurrency control and reliability of a DDBMS. A given database D can be allocated to a network by using one of the following strategies:

1. D is allocated to a central node NC
2. D is uniquely allocated to a node N where it is most frequently accessed
3. D is allocated to N and a duplicate copy of D is kept at the central node NC
4. D is allocated to N and duplicate copies of D are kept at an arbitrary set of nodes N1, N2, N3, etc.
5. D is allocated to every node (replicated)
6. D is partitioned into D1, D2, D3 which are allocated by using the strategies 1 to 5.

The cost/benefit of this decision can be estimated in terms of storage cost, communication costs (cost to read, cost to update), response time and data availability. Fig. 5.1 briefly illustrates the tradeoff by showing the data duplication on x axis and costs on y axis. It can be seen from Fig. 5.1 that:

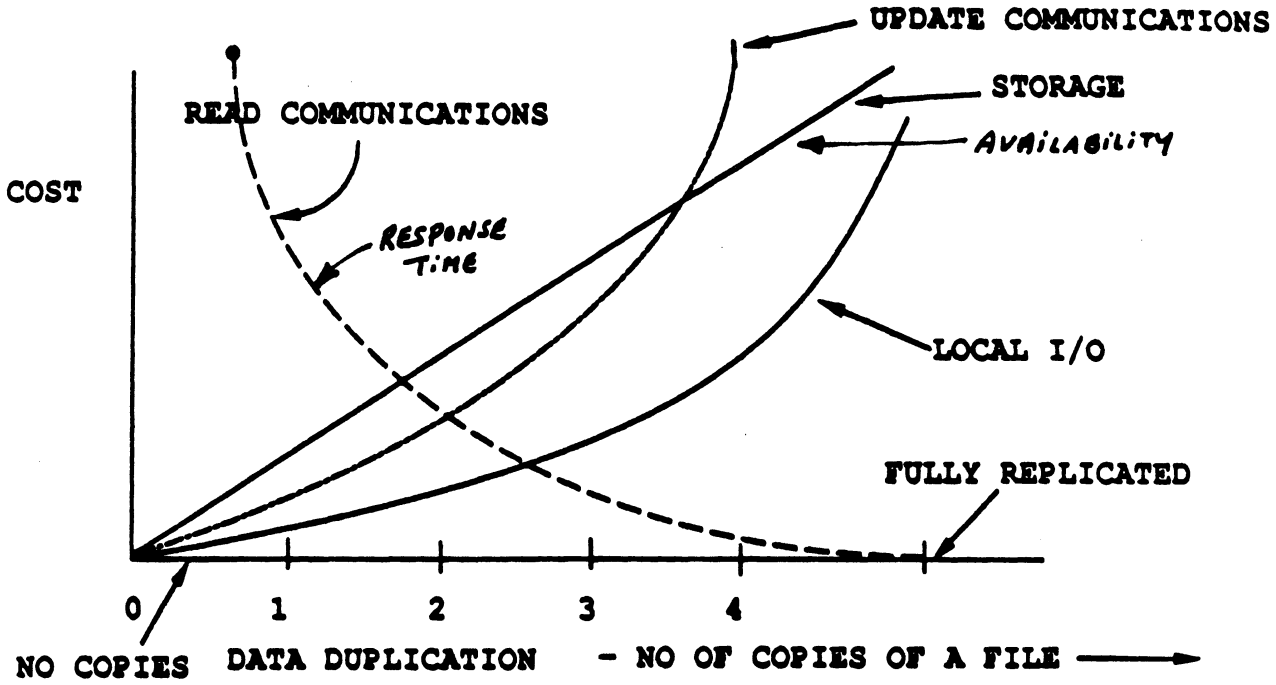
1. the storage cost increases as the number of copies increases

2. the read communication cost decreases as the number of copies increases because most data can be found at local nodes thus eliminating need for communication calls

3. the update communication cost increases with the number of copies because duplicated data will need to be updated

4. the availability of data increases with the number of copies in the system

Figure 5.I: **TRADEOFF IN DATA ALLOCATION**



- OPTIONS:
  - * UNIQUE ALLOCATION: ASSIGN FILE TO ONE NODE
    - · AMOUNT OF STORAGE: SMALL
    - · READ COMMUNICATIONS: HIGH
    - · UPDATE COMMUNICATION: SMALL
    - · LOCAL I/O AT EACH NODE: SMALL
  - * DUPLICATE ALLOCATION: MORE THAN ONE COPY
    - · AMOUNT OF STORAGE: LARGE
    - · READ COMMUNICATIONS: SMALL
    - · UPDATE COMMUNCATION: HIGH
    - · LOCAL I/O AT EACH NODE: HIGH

An optimal data allocation can be theoretically determined which minimizes the total cost (storage + communication + local I/O) subject to some response time and availability constraints. This problem, traditionally referred to as the File Allocation Problem (FAP), in computing networks was first addressed by Wesley Chu in 1969 <Chu69>. Since then over 20 different file allocation algorithms have appeared in literature <Bryand81, Bucci79, Buckles79, Casey72, Chandy76, Chang81, Chu69, Coffman80, Doty82, Eswaran74, Finckenscher84, Fisher80, Irani81, Khabbaz79, Kimbleton77, Koh85, Levin75, Mahmood76>. Earlier allocation problems were simple, while more recent methods are actual design methodologies which utilize the allocation techniques as one of the decisions <Umar84, Jain87, Ceri87>. A common FAP is formulated as follows:

. Given:
  - files f1, f2, f3,,,
  - Nodes n1, n2, n3,,,,,
. Determine:
  - ADN(f,n) = 1 if f is allocated to n, 0 otherwise
. To minimize:
  - storage cost + communication cost + local processing cost
. Subject to:
  - response time constraints
  - availability constraints
  - network topology
  - security restrictions

Variants of this problem are cast into a mathematical programming problem and are solved by using nonlinear, integer and/or dynamic programming methods.

Application of the existing FAP problems depends on the nature of the problem, the availability of information needed to reach an exact solution and the need to determine optimal versus approximate solutions in real life. Based on past experience in the distributed database allocations, it has been found that for several real life situations, sophisticated FAP are needed rarely. In most cases, several data allocation decisions can be made by excercising judgement and using real life constraints of security and management. However, it is preferable to use simple analytical models to support the decisions and improve insights. A method which combines heuristics with formal models to allocate data in DDBMS is described in section 5.4.

5.2 DATABASE DEFINITION

After a database has been allocated, the views (schema) of the database have to be defined. The following three levels of schema definitions are based on the ANSI/X3/SPARC recommendations for data description:

1. Local (internal) schema which show how the data is stored on each node. The format of the internal schema is dependent on the LDBMS of each node.

2. A global (conceptual) schema which describes the data throughout a network and shows what data is at what node. In case of heterogeneous LDBMS, the global schema format is different from the internal schema format. The global schema is usually stored in a global directory.

3. A user (external) schema which shows how a user will view and manipulate the data.

The problem of schema design is straightforward if all the LDBMS are homogeneous but is of critical importance in a network with heterogeneous databases. At present most DDBMS are using SQL for schema definition and manipulation <Gligor83, Rauch-Hinden87>. The interested reader is referred to <Cardenas87> for schema design considerations for heterogeneous DDBMS.

5.3 DATABASE ACCESS AND MANIPULATION

The user queries (DML) are written by using the external schema. If one homogeneous query language, like SQL, is used throughout the network, then the user of a DDBMS should not see any difference between a DDBMS and a local DBMS. However, if different DML are supported by the DDBMS, then the main user-oriented issue is: what DML should be used from what node? This issue is not discussed here in detail.

5.4 A DATABASE DESIGN PROCEDURE

A simple database design procedure is suggested in Fig. 5.2 and an analytical model to support the procedure is given in Table 5.1. This procedure and model are presented here so that a potential designer of DDBMS can use a quick paper and pencil method to compute the costs for data allocations. The steps of the procedure are:

Determine the database requirements, by interviewing end-users, and normalize them, say in the third normal form. Let D1, D2,,,, D50 be the database relations generated. Define a global schema of the database.

2. Partition/cluster relations into:

   - location specific data which is primarily used at one site - assume D1,,,,D40 are location specific due to security or other situations (robotics, payroll, etc)

   - location independent data which may be accessed from many sites - assume D41,,D50 are location independent

3. Allocate location specific data to nodes where it is needed most frequently and eliminate it from further considerations

4. Reduce the number of allocation independent files due to:

   - security: assume D41-D43 must be restricted to one node due to security restrictions

   - management: assume D44-D45 must be allocated to a central site for management control

5. Allocate the rest of the files: D46-D50 by using a FAP by using the following tradeoffs:

   . amount of storage
   . read communications (cost, time)
   . update communication (cost, time)
   . local I/o at each node

Fig. 5.2: A Distributed Database Design Procedure

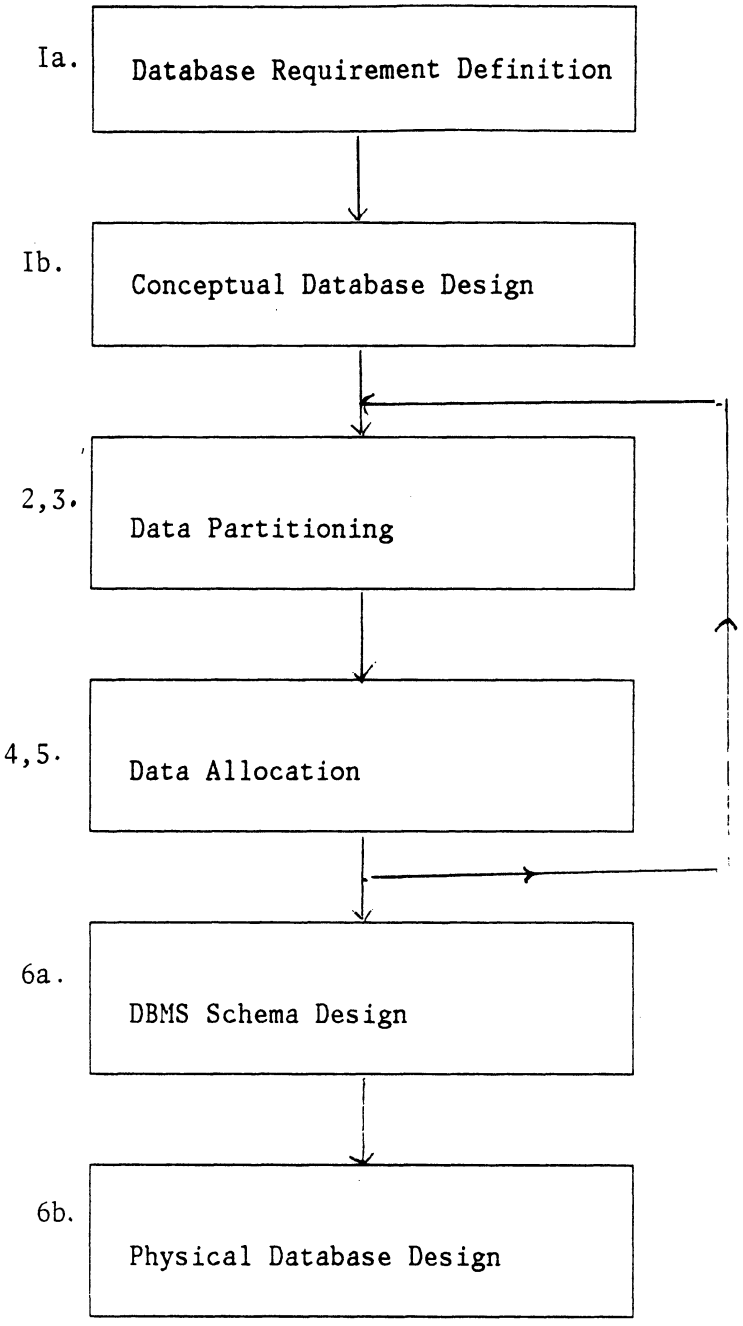


TABLE 5.1 SIMPLE ANALYTICAL MODEL

. ALLOCATE A DATASET d TO n NODES:
.CALCULATE STORAGE, READ/UPDATE COMMUNICATION, LOCAL I/O
.CALCULATE TIME, COST, ETC.

1. STORAGE OCCUPIED BY d- $STORE(d)=\sum_{i} FSIZE(d).A(d.,i)$

WHERE FSIZE(d): SIZE OF d IN BYTES
A(d,i) = 1 IF d IS ALLOCATED TO NODE i, 0 OTHRWISE
STORE(d)= TOTAL STORAGE OCCUPIED BY d IN NETWORK

2. NO. OF UPDATES U(i,j,d)= UP(i,d).A(d,j), 0 IF i=j
WHERE U(i,j,d): N0. OF UPDATES ISSUED BETWEEN i AND j FOR d
UP(i,d)= NO. OF UPDATES ISSUED FROM i TO DATASET d

3. NO. OF RETRIEVALS R(i,j',d)= RD(i,d).A(d,j')
j' IS THE CLOSET NODE, R(i,j',d)=0 IF i=j'
WHERE R(i,j,d) : NO. OF READS ISSUED BETWEEN i AND j FOR d
RD(i,j')= NO. OF READS ISSUED FROM i TO DATASET d

4. NO. OF LOCAL I/O AT j=LIO(j,d)= $\sum_{i} U(i,j,d) + \sum_{i} R(i,j,d)$

5. CAN CALCULATE OTHER PARANETERS:

. TOTAL STORAGE COST= $\sum_{i} STORE\ (d)$

. RESPONSE TIME (WITHOUT QUEUING)

$$= \sum_{j'}\sum_{i} R(i,j',d).SPEED(i,j) + \sum_{j}\sum_{i} U(i,j,d).\ SPEED(i,j) + \sum_{j} LIO\ (j,d).IOSPEED(j)$$

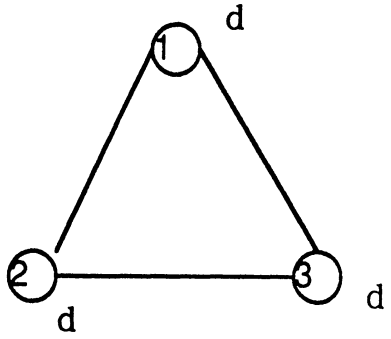
EXAMPLE: ALLOCATE d TO 3 NODES: FSIZE = 100K

. A=1 0 1 , UP= 10 15 0 ,RD= 10 15 20
1. STORAGE= 100X 1+ 100X 0+100 X 1=200k
2. UPDATE = U(i,j,d) =

	1	2	3
1	0	0	10
2	15	0	15
3	0	0	0

3. READS = R(i,j,d) =

	1	2	3
1	0	0	0
2	15	0	0
3	0	0	0



(can use analytical/simulation for analysis)

6. Modify the global schema to show the data allocations

7. Conduct a local database design for each of the databases and define the external user schema.

The simple analytical model shown in Table 5.1 can compute the storage occupied, communication traffic and local processing traffic in terms of data allocations. These dependent variables can be used to compute rough estimates of response time and availability. An example illustrates this model. This model can be easily translated to a Fortran or Pascal routine. A more sophisticated routine to calculate the traffic and the response time is given in <Umar84, Umar85>.

6. DDBMS IMPLEMENTATION CONSIDERATIONS

A DDBMS can be implemented by using a variety of approaches in data allocations, directory definition and allocation, concurrency control algorithms, transaction processing strategies and recovery management. The choice of right techniques and algorithms depends on the facilities to be supported by the DDBMS. Fig. 6.1 shows a framework to help a potential developer in making this decision. The framework uses the following three variables as the constrained variables:

- Homogeneous versus heterogeneous LDBMS
- Network control (central, distributed) where one node is designated to coordinate the interactions between all other nodes
- Number of copies allowed in a network, i.e. single, two copies (one at central site, one at local site), and more than two

Based on these constrained variables, several options are defined. For each option, the choices available in terms of database definition, directory allocation, concurrency control, transaction processing, and recovery management are listed. It should be noted that the simplest implementation of a DDBMS is for option 1 (homogeneous, centrally controlled, unique data allocation) and the most difficult is for option 5 when the DBMS are heterogeneous, there is no central control, and the data can be replicated.

Several DDBMS are being announced for commercial use this year <Graham87, Rauch-Hinden87, McCord87>. These systems implement different facilities. Some of the commercially available DDBMS are (see <Rauch-Hinden87> for detailed description):

- ADR/Dnet of Applied Data Research
- Cincom's DDBMS
- Shard, Adaplex and Model 204 of Computer Corporation of America
- cXDMS of Computer-X Inc.
- IDMS/Distributed of Cullinet
- VAX Data Distributer of Digital Equipment Corp.
- Starburst of IBM
- Informix-Star of Informix Software
- SQL*STAR of Oracle, Inc.
- Ingres/Star of Relational Technology
- Sybase/Star Server of Sybase, Inc.
- Encompass of Tandem Computers

Unify's DDBMS

Several implementations support option 1 shown in Fig. 6.1. For example, most systems use SQL as a standard query language, assume some level of centralized control and allow limited data distribution. Several home grown implementations allow centralized data with distributed skeleleton files. The approach of remote procedure call is supported by CICS-ISC and Local area network file servers like Btrieve on Token Ring. This approach allows a single application to access many databases, i.e. program P can access file F1 on node n1 and F2 on node F2. This approach allows only one copy of a file.

Option 2 is supported by DDBMS where the central site database is periodically extracted and sent to user sites. The central site database can be retrieved and updated; user site data can be retrieved but cannot be updated. An example of this approach is the DEC VAX Data Distributor. In this DDBMS, a user extracts and replicates relational RDB/VMS which are sent to user nodes via DECNET.

Options 3 and 4 are supported by some DDBMS; a database is subdivided (partitioned), say by region or product, and then the data is replicated while a user views one logical database. An example of this approach is the Tandem Computer's Encompass which is one of the oldest DDBMS. Encompass allows distributed queries, distributed updates and synchronized replicated data. Relational Technology's INGRES/STAR and Oracle's Oracle SQL*STAR are other examples of this type of implementation. Both can retrieve data, perform joins, and update data from different sites. Current versions restrict update to one site. Other examples of implementations in progress in these options are ADR'S DNET 2.0, CCA's ADAPLEX, distributed MODEL 204, UNIFY CORP's distributed version of Unify DBMS and Informix Software Incs distributed INFORMIX DBMS.

An improvement over the partitioned support are the fragmentation systems where horizontal and vertical fragments of a database are allocated to different sites. The user does not know that the database has been fragmented. Relational Technology Inc. announced fragmentation support for late 1988. No other significant plans to support fragmentation have been announced. An area of significant activity at present is the distribution among micros, minis and mainframes. Details of these efforts can be found in the papers by Rauch-Hinden <Rauch-Hinden87>.

Some of the oldest implementations of DDBMS, e.g. the Tandem Encompass, are described in detail by Gray <Gray87>. Cohan et al <Cohan82> describe a DDBMS that has been implemented for network services. In addition to the commercially available DDBMS, several research and development DDBMS are being developed in different parts of the world. Examples of these systems are:

- COSYN being developed in France
- CSIN being developed in the US
- Sirius-Delta being developed in France
- Sirius-POLYPHEME being developed in France
- Distributed DPLS being developed in Japan
- XNDM being developed in the US
- Multibase being developed in the US
- JIPDEC-Distributed DBMS (Japan)
- POREL (Germany)
- Nixdorf (Germany)

Details of these implementations can be found in <Gligor83>.

FIG.6.1: DDBMS IMPLEMENTATION OPTIONS

CASE 1: HOMOGENEOUS, CENTRALLY CONTROLLED, NO DATA DUPLICATION
- a. DATABASE DEFINITION AND TRANSLATION: NOT DIFFICULT
- b. DIRECTORY ALLOCATION: AT INDIVIDUAL SITES
- c. DATABASE CONSISTENCY/CONCURRENCY: NOT NEEDED FOR UNIQUE DATA
- d. QUERY OPTIMIZATION: NOT DIFFICULT
- e. FAILURE HANDLING: NOT DIFFICULT

CASE 2: SAME AS 1, COPY CAN EXIST AT CENTRAL SITE
- a. DATABASE DEFINITION AND TRANSLATION: NOT DIFFICULT
- b. DIRECTORY ALLOCATION: MAY BE AT CENTRAL SITE
- c. DATABASE CONSISTENCY/CONCURRENCY: CAN BE BASIC 2PL
- d. QUERY OPTIMIZATION: SIMPLE HILL CLIMBING
- e. FAILURE HANDLING: SIMPLE TWO PHASE COMMIT

CASE 3: SAME AS 2, SEVERAL COPIES CAN EXIST
- a. DATABASE DEFINITION AND TRANSLATION: NOT DIFFICULT
- b. DIRECTORY ALLOCATION: MAY BE AT CENTRAL SITE
- c. DATABASE CONSISTENCY/CONCURRENCY: CENTRAL CONTROLLED
- d. QUERY OPTIMIZATION: MANY CHOICES
- e. FAILURE HANDLING: TWO PHASE COMMIT

CASE 4: SAME AS 3, NOT CENTRALLY CONTROLLED
- a. DATABASE DEFINITION AND TRANSLATION: NOT DIFFICULT
- b. DIRECTORY ALLOCATION: MANY CHOICES
- c. DATABASE CONSISTENCY/CONCURRENCY: MANY CHOICES
- d. QUERY OPTIMIZATION: MANY CHOICES
- e. FAILURE HANDLING: MANY CHOICES

CASE 5: SAME AS 4, HETEROGENEOUS
- a. DATABASE DEFINITION AND TRANSLATION: VERY DIFFICULT, MANY CHOICES
- b. DIRECTORY ALLOCATION: MANY CHOICES
- c. DATABASE CONSISTENCY/CONCURRENCY: MANY CHOICES
- d. QUERY OPTIMIZATION: MANY CHOICES
- e. FAILURE HANDLING: MANY CHOICES

7 SUMMARY AND CONCLUSIONS

Design and implementation of DDBMS involves many challenges and choices. This report has attempted to present four different aspects of DDBMS. First a set of requirements for a generalized DDBMS are presented to provide the scope of issues to be handled by a DDBMS. Second, a reference architecture is presented which clusters the main issues into several modules. The architecture is used to explain the issues encountered and approaches available in database definition, concurrency control, query processing and recovery/failure management. Third, the usage aspects of DDBMS, e.g. the distributed database design problem, are explained and a simple design method is presented. Fourth, the implementation issues are summarized and various commercially available DDBMS are discussed.

The following general observations can be made:

1. The complexity of the DDBMS to be developed depends on the options supported. It is simple to develop DDBMS which support single copy and homogeneous LDBMS approach.

2. DDBMS have moved from research to commercial availability. However, it is difficult to determine what approaches are being adopted by commercial DDBMS.

3. Evaluation of commercial DDBMS is difficult due to the vagueness between promised versus available facilities.

4. Tandem system has one of the oldest and proven systems due to its architecture which supports a large number of small computers.

5. SQL is considered a standard for transparency, but:
   - SQL conversion to local DBMS is not trivial
   - all SQL are not same, e.g. ISO SQL is a subset of DB2 SQL
   - relational systems may not be adequate for engineering applications like CAD/CAM

In short, the difficulties encountered in implementing DDBMS must be carefully weighed against the advantages of DDBMS over centralized systems. For several applications, it may still be better to provide a centralized database or use a single copy DDBMS after all of the costs for query processing, concurrency control and failure management have been taken into account.

REFERENCES

1. Bernstein, P. A. and Goodman, N., "Concurrency Control in Distributed Database Systems", ACM Computing Surveys, Vol.13, No.2, June 1981, pp.185-222.

2. Bernstein, P. A. and Chiu, D.W., "Using Semi-Joins to Solve Relational Queries", JACM, Jan. 1981.

3. Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and and Recovery in Database Systems", Addison Wesley, 1987

4. Bryand, R.M., and Agre,J.R., "A Queuing Network Approach to the Module Allocation Problem in Distributed Systems", ACM Conf. on Measurement and Modelling of Computer Systems, Sept. 1981, pp. 181-190.

5. Bucci,G., Streeter,D.N.,: "A Methodology for the Design of Distributed Information Systems", CACM,v.22, no.4, April 1979, pp. 233-245.

6. Buckles, B.P. and Harding, D.M., 'Partitioning and Allocation of Logical Resources in a Distributed Computing Enirnment', General Research Corporation Report, Huntsville, Alabama, 1979.

7. Cardenas, A.F., "Heterogeneous Distributed Database Management: the HD-DBMS", Proceedings of the IEEE, May 1987, pp.588-600.

8. Casey, R.G.,"Allocation of Copies of a File in an Information Network", SJCC 1972, AFIPS Press, Vol. 40, 1972

9. Ceri, S., Pernici, B., and Wiederhold, G., "Distributed Database Design Methodologies", Proceedings of the IEEE, May, 1987, pp.533-546.

10. Chandy, D.M. and Hewes, J.E. 'File Allocation in Distributed Systems', Proc. of the Intl Symp on Computer Performance Modelling, Measurement and Evaluation, March 1976, pp.10-13.

11. Chang, S.K. and Liu, A.C., ' A Database File Allocation Problem', COMPSAC, 1981, pp.18-22.

12. Chu, W.W., 'Optimal file allocation in a multiple computer system', IEEE Tran. on Computers, oct. 1969, pp. 885-889.

13. Chu, W.W., "Performance of File Directory Systems for Data Bases in Star and Distributed Networks", 1976 NCC, Vol. 45, 1976.

14. Chu, W.W., "Distributed Data Bases", Handbook of Software Engineering, Edited by C.R. Vick and C.V. Ramamoorthy, Van Nostrand REinhold, 1984

15. Ceri, S., and G. Pellagitti, "Distributed Databases: Principles and Systems", McGraw Hill, 1984.

16. Ceri, S., Pernici, B., Wiederhold, "Distributed Database Design Methodolgies", Proceedings of the IEEE, May 1987, pp.533-546.

17. Coffman, E.G., Gelenbe, E, et al, 'Optimization of the Number of Copies in Distributed Databases', Proc. of the 7th IFIP Symposium on Computer Performance Modelling, Measurement and Evaluation , May 1980, pp. 257-263.

18. Cohen, D., Holcomb, J.E., and Suryanarayana, M.B., "Distributed Database Management to Support Network Services", IEEE Conference on Communications, 1982.

19. Date, C.J., "An Introduction to Database Systems", Fourth edition, Vol.1 and Vol. 2, Addison Wesley, 1986

20. Date, C.J., "A Guide to DB2", Addison Wesley, 1984

21. Decitre, P., "A Model for Describing Distributed Database Management System Architecture", 16th Annual IEEE Electronics and Aerospace Systems Conference", Sept. 1983.

22. Doty, K.W., McEntyre, P.L. and O'Reilly, J.G., 'Task Allocation in a Distributed Computer System', Proc. of IEEE INFOCOM, 1982, pp. 33-38.

23. Epstein, R., Stonebraker, M., Wong, E., "Distributed Query Processing in a Relational Database System", ACM SIGMOD, Austin, TX, 1978

24. Epstein, R., "Query Optimization: A Game of Time and Statistics", Unix Review, May 1987, pp. 28-29.

25. Eswaran, K.P; Gray,J.N, et al "Notions of Consistency and Predicate Locks in a Database System", CACM,Vol.19, No.11,Nov.1976

26. Eswaran, K.P; "Allocation of Records to Files and Files to Computer Networks", IFIP, 1974, pp.304-307.

27. Farrag, A.A., and Ozsu, M.T., "Towards a General Concurrency Control Algorithm for Database Systems", IEEE Trans. on Software Engineering, Vol. SE-13, No.10, Oct. 1987, pp.1073-1073.

28. Finckenscher, G. 'Automatic Distribution of Programs in MASCOT and ADA Environments', Royal Signal and Radar Establishment, London, 1984.

29. Fisher, P., Hollist, P. and Slonim, J. "A Design Methodology for Distributed Databases", Proc. IEEE Conf. on Distributed Computing, pp.199-202, Sept.1980.

30. Fisher, M.L. and Hochbaum, D.sp., "Database Location in Computer Networks", ACM Journal, V.27, N.4, Oct. 1980.

31. Garcia-Molina, H., "Performance of Update Synchronization Algorithms For Replicated Data in a Distributed Database", Ph.D. Dissertation, Stanford University, June 1979.

32. Garcia-Molina, H., Abbot, R.K., "Reliable Distributed Database Management", Proceedings of the IEEE, Vol. 75, No. 5, May 1987, pp601-620.

33. Gligor, V.D., and Fong, E., "Distributed Database Management Systems: An Architectural Perspective", Journal of Telecommunication Networks, Vol. 22, No. 3, pp.247-270, Fall 1983.

34. Graham, G., "Real-World Distributed Databases", Unix Review, May 1987.

35. Gray, J., "Notes on Datbase Operating Systems", in Operating Systems: An Advanced Course, Springer-Verlag, N.Y. 1979, pp.393-481.

36. Gray, J., "The Transaction Concept: Virtues and Limitations", Proceedings of Conference on Very Large Databases, Sept. 1981, pp. 144-154.

37. Gray, J., "Transparency in Its Place", Unix Review, May 1987.

38. Gray, J.N., and Anderson, M., "Distributed Computer Systems: Four Cases", Proceedings of the IEEE, May 1987, pp.719-729.

39. Goodman, N., et al., "Query Processing in SDD-1: A Ssystem for Distributed Databases", CCA, Technical Report, CCA-79-06, 1979.

40. Goodman, N., "Interview with Phil Bernstein", Unix Review, May 1987. GYLY76). Gylys, V.B. and Edwards, J.A, 'Optimal Partitioning of Workload for Distributed Systems', COMPCON, fall 1976.

41. Hebalkar, P.G. 'Logical Design Considerations for Distributed Database Systems', IEEE COMPSOC, Nov. 1977, pp.562-580.

42. Rauch-Hinden, W., "True Distributed DBMSes Presage Big Dividends", Mini-Micro Systems, May and June, 1987

43. Heinselman, R.C. "System Design Selection for Distributed databases", Proc. IEEE Conf. on Distributed Computing, pp.203-210, Sept.1980.

44. Hevner, A.R., and Yao, S.B., "Query Processing in Distributed Database Systems", IEEE Transactions on Software Engineering, Vol. SE-5, No. 3, May 1979.

45. Hevner, A.R., 'A Survey of Data Allocation and Retrieval Methods for Distributed Systems', School of Business and Management Working paper # 81-036, Univ. of Maryland, Oct. 1981.

46. Hevner, A.R., and Yao, S.B., "Querying Distributed Databases on Local Area Networks", Proceedings of the IEEE, May 1987, pp.563-572.

47. IDMS System Overview, Westwood, Mass., Cullinane Database Systems, 1981.

48. Irani, K.B. and Khabbaz, N.G., 'A Combined Communication Network Design and File Allocation for Distributed Databases', 2nd Intl Conf on Distributed Systems, Paris, April 1981.

49. Jain, H.K., "A Comprehensive Model for the Design of Distributed Computer Systems", IEEE Trans. on Software Engineering, Vol. SE-13, No.10, Oct. 1987, pp.1092-1105.

50. Jarke, M. and Koch, J., "Query Optimization in Database Systems", ACM Computing Surveys, June 1984, pp.111-152.

51. Kapp, D., and Leben, J., "IMS Programming Techniques", Van Nostrand-Reinhold Company, 1978.

52. Khabbaz, G.N, "A Combined Network Design and File allocation in Distributed Computer Networks", Ph.D dissertation proposal, Univ. of Michigan, 1979.

53. Kimbleton,S.R; "A Fast Approach To Network Data Assignment", Proc. of 2nd Berkeley Workshop on Distributed Data Management and Computing Networks, May 1977, pp.245-255

54. Koh, K. and Eom, Y.I "A File Allocation Scheme for Minimizing the Storage Cost in Distributed Computing Systems", First Pacific Computer Communication Symposium, Seoul, Korea, Oct. 1985, pp.310-317.

55. Kohler, W. H., "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems", ACM Computing Surveys, Vol.13, No.2, June 1981, pp.149-184.

56. Kroenke, D., "Database Processing", SRA, 1977.

57. Larson, J. A., "A Flexible Reference Architecture for Distributed Database Management", Proceedings of ACM 13th Annula Computer Science Conference, March, 1985, New Orleans, pp.58-72.

58. Larson, B., "A Retrospective of R*: A Distributed Database Management System", Proceedings of the IEEE, May 1987, pp.668-673.

59. Levin, K.D and Morgan, L.H. " Optimizing Distributed Data Bases- A Framework For Research", Proc. NCC, 1975, Vol.44, pp.473-478

60. Lim, P.A., "CICS/VS Command Level with Ans Cobol Examples" Van Nostrand Reinhold, 1982.

61. Lu, P.M. and Yau, S.B.,"A Methododlogy for Representing the Formal Specifications of Distributed Computer System Software Design", Proc. IEEE Conf. on Distributed Computing Systems, pp.31-42, Oct. 1979.

62. Mahmood, S. and Riordan, J.sp.; "Optimal Allocation of Resources in Distributed Information Networks", ACM Trans. on Database systems, Vol.1, No. 1, March 1976, pp. 66-78

63. Mariani, M.P. and Palmer, D.F. (eds.), "Tutorial: Distributed System Design', EHO 151-1, IEEE Computer Society, 1979.

64. Mariani, M.P. and Palmer, D.F "Software Development for Distributed Computing Systems", Handbook of Software Engineering, edited by C.V.Ramamoorthy and C.R.Vick, Van Nostrand, 1984, pp.656-674.

65. Martin, D., "Advanced Database Techniques", MIT Press, 1986.

66. McCord, R. and Hanner, M., "Connecting Islands of Information", Unix Review, May 1987.

67. Rothnie, J.B. and Goodman, N. "A Survey of Research and Development in Distributed Data Base Management", 3rd Conference on Very Large Data Bases, Tokyo, Oct. 1977, pp.48-60

68. Sacco, G.V. and Yao, S.B., "Query OPtimization in Distributed Databases", Working Paper MS/S #81-029, College of Business Administration, University of Maryland, 1981.

69. Shatz, S.M., and Jia-Ping Wang, "Introduction to Distributed-Software Engineering", IEEE Computer, October, 1987, pp.23-32.

70. Stamper, D., "Business Data Communications", Benjamin/Cumings, 1986.

71. Stone, H., "Parallel Querying of Large Databases: A Case Study", IEEE Computer, October, 1987, pp.11-21.

72. Stonebraker, M., "Concurrency control and consistency of multiple copies of data in distributed Ingres", IEEE Trans. on Software Engineering, SE-5, 3, (May 1979), pp.188-199.

73. Svobodova, L., "File Servers for Network-Based Distributed Systems", ACM Computing Surveys, Dec. 1984, pp.353-398.

74. Tanenbaum, A.S, and van Renesse, R. "Distributed Operating Systems", Computing Surveys, pp. 419-470, December 1985.

75. Tanenbaum, A.S., "Computer Networks", Prentice Hall, 1981.

76. Teorey, T.J and Fry, J.P "Design of Database Structures", Prentice Hall, 1982.

77. Ullman, J., "Principles of Database Systems", John Wiley, 1982.

78. Umar, A "The Allocation of Data and Programs in Distributed Data Processing Environments", Ph.D. Dissertation, Univ. of Michigan, 984. Also published as Computing Research Laboratory report CRL-TR-22-84, University of Michigan, Ann Arbor, Michigan.

79. Umar, A. and Teorey, T.J "A generalized approach to Program and Data Allocation in Distributed Systems", First Pacific Computer Communication Symposium, Seoul, Korea, Oct. 1985, pp.462-472.

80. Vasta, J., "Understanding Database Management Systems", Wardsworth, 1985.

81. Wilbur, S., and Bacarisse, "Building Distributed Systems with Remote Procedure Call", Software Engineering Journal, Sept. 1987, pp.148-159.

82. Walpole, J. et al, "Transaction Mechanisms for Distributed Programming Environments", Software Engineering Journal, Sept. 1987, pp.169-171.

83. Wolfe, P.E., "Computer Aided Design Report", Vol. 7, No. 4, April. 1987.

84. Wolfson, O., "The Overhead of Locking (and Commit) Protocols in Distributed Databases", ACM Trans. on Database Systems, Vol. 12, No. 3, Sep. 1987, pp. 453-471.

85. Wong, E., "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases", Berkeley Workshop on Distributed Data Management and Computing Networks, 1977.

86. Woodside, C.M and Tripathi, S.K "Optimal Allocation of File Servers in a Local Area Network Environment", IEEE Trans. on Software Engineering, Aug. 1986, Vol. SE-12, No. 8, pp. 844-848.

87. Yao, S.B., "Optimization of Query Evaluation Algorithms", ACM Trans. on Database Systems, Vol. 4, No. 2, June 1979, pp.133-155

88. Yu, C., and C. Chang, "Distributed Query Processing", ACM Computing Surveys, vol. 16, no. 4, pp.399-433, Dec. 1984.