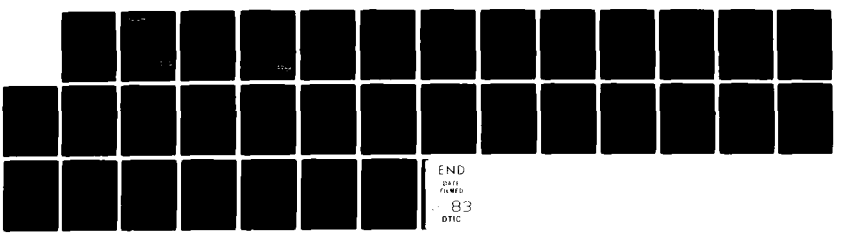
AD-A125 163 AN OVERVIEW OF THE PRELIMINARY DESIGN OF SDD-1: A SYSTEM FOR DISTRIBUTED DATABASES(U) COMPUTER CORP OF AMERICA CAMBRIDGE MA J B ROTHNIE ET AL. 31 MAR 77 1/1

UNCLASSIFIED CCA-77-04 N00039-77-C-0074 F/G 9/2 NL

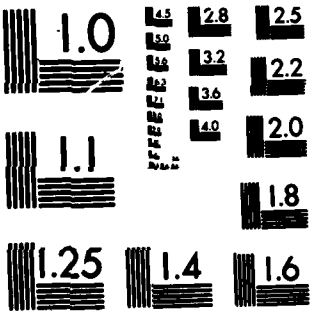


END

DATE FILMED

83

DTIC



An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases

Technical Report
CCA-77-04
March 31, 1977

J. B. Rothnie
N. Goodman

CONTRACT N00039-77-C-0074

Computer Corporation of America

Abstract

SDD-1, A System for Distributed Databases, is currently being designed and implemented by Computer Corporation of America. SDD-1 is composed of a collection of datamodules which may be dispersed geographically and which are assumed to communicate over channels which may vary in bandwidth and delay. Also, the system supports redundant databases, meaning that portions of databases may be stored at two or more datamodules in order to improve the reliability and efficiency of database operations.

This paper presents a brief overview of several key facets of SDD-1. These include:

- the system architecture,
- the data distribution concepts used by the system,
- SDD-1's approach to directory management,
- the method employed for efficiently handling updates to redundantly stored data, and
- SDD-1's method for efficiently retrieving data which is dispersed at distant datamodules.

Table of Contents

1. Introduction

SDD-1 is a distributed database system being designed and implemented by Computer Corporation of America (CCA) in a project sponsored by the Advanced Research Projects Agency (ARPA) of the Department of Defense. Work on the system is currently in progress. This paper describes the system's preliminary design.

SDD-1 is designed to support databases distributed worldwide over hundreds of database sites. The database sites are assumed to communicate over heterogeneous communication channels which may vary in bandwidth and delay, and it is not assumed that all sites are able to maintain continuous communication with each other. Also, SDD-1 is designed to support databases which are stored redundantly, meaning that some or all logical data items can be stored at multiple database sites in order to enhance the reliability and responsiveness of the system.

The objectives which SDD-1 is intended to achieve are the following:

1. reliability/survivability - the system must continue operation despite the failure or inaccessibility of one or more of its database sites.

2. "tunable" efficiency - it must be possible to distribute data in such a manner that portions which are heavily used in a given geographical region can be stored near that region.

3. modular upward scaling - as databases increase in size and usage, it must be possible to augment system capacity to accommodate these increases by the incremental addition of new database sites.

It is important to note the key role which data redundancy plays in achieving these objectives. Reliability and survivability are enhanced since SDD-1 can continue to access critical data items even if some of the database sites at which the data are stored become inaccessible. Efficiency is enhanced since data items which are accessed by widely separated user communities can be stored nearby <u>each</u> of the communities. Redundancy also enhances scalability since growth in database usage can be accommodated by increasing the number of data copies rather than by increasing the speed of memory units.

On the other hand, data redundancy introduces severe problems in performing updates in a consistent manner. The approach taken by SDD-1 in handling these redundant update problems is addressed in Section 4 of this paper and in [ROTHNIE and GOODMAN], [ROTHNIE et al], and [BERNSTEIN].

Other distributed database system designs (e.g., [ALSBERG and DAY], [ELLIS], and [THOMAS-b]) have permitted redundantly stored data, but have required that the entire database be stored at every database site. This restriction is unrealistic for large databases and effectively precludes "tunable" efficiency and upward scalability.

This paper summarizes certain key characteristics of the SDD-1 design. Section 2 describes the architecture of the system and presents the framework used by SDD-1 for defining the distribution and redundancy of logical databases. Sections 3 - 5 then describe the manner in which SDD-1 deals with key technical difficulties in distributed database management: Section 3 discusses the management of directory information; Section 4 explains the approach taken in SDD-1 to the problem of updating redundantly stored data; and Section 5 presents SDD-1's approach to efficiently processing retrievals involving dispersed data.

2. Architecture and Distribution Concepts

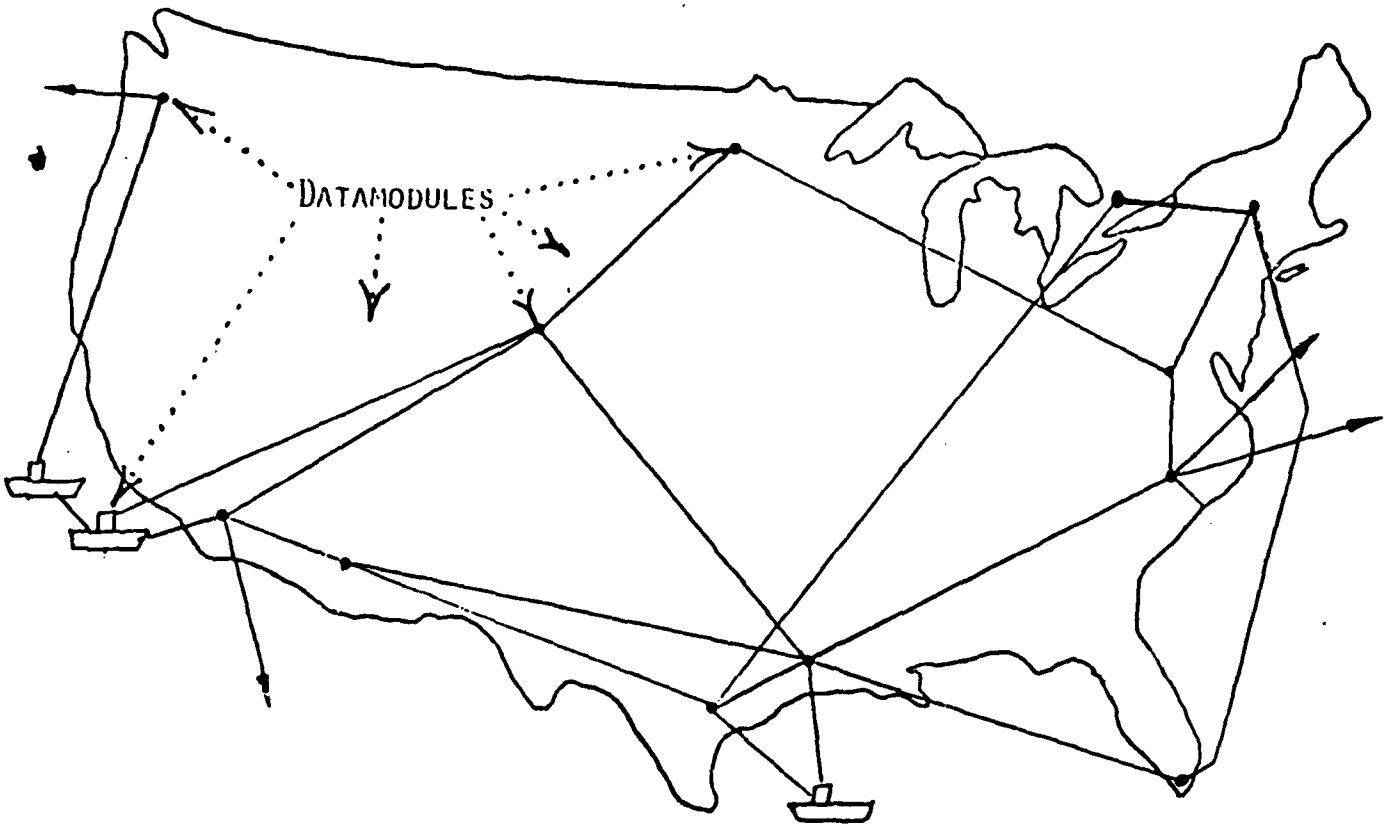
2.1 Global System Architecture

Figure 2.1 illustrates the general architecture of SDD-1. The system is composed of a set of components called datamodules which communicate over assorted communication channels. The datamodules are all functionally identical; that is, they respond identically to external stimuli. However, there may be a great variety of datamodule implementations. For example, some datamodules may support mass memories and operate as major data repositories in the net while others may be small systems with little storage, operating as data caches for quick response to nearby users.

The distribution and redundancy of data in SDD-1 is invisible to users. A user connecting to any datamodule is presented with the illusion that a complete and non-redundant database is resident at that datamodule. In response to users' queries SDD-1 takes responsibility for

General System Architecture of SDD-1
Figure 2.1

SDD-1 GENERAL SYSTEM DESCRIPTION



- SDD-1 COMPRISED OF DISTRIBUTED DATAMODULES.
- DISTRIBUTION IS INVISIBLE TO USERS.
- DATA IS STORED REDUNDANTLY.

locating the desired data among the distributed network of datamodules and for updating all copies of changed data items; ideally users are able to interact with the distributed system as easily as with a conventional, centralized one.

The objective of relieving users of the need to be aware of distribution issues is similar to goals pursued in the distributed operating systems being developed today, e.g. by the National Software Works (cf [SCHANTZ and MILLSTEIN]), the National Bureau of Standards NAM project (cf [ROSENTHAL]), and the Arpanet RSEXEC project (cf [THOMAS-a]). These projects take a collective computing resource distributed geographically on a computer network and make it available to users in an integrated, easy to use manner. This approach in the distributed database area has also been discussed by [ALSBERG et al], [ALSBERG and DAY], [ELLIS], [STONEBRAKER and NEUHOLD], and [THOMAS-b].

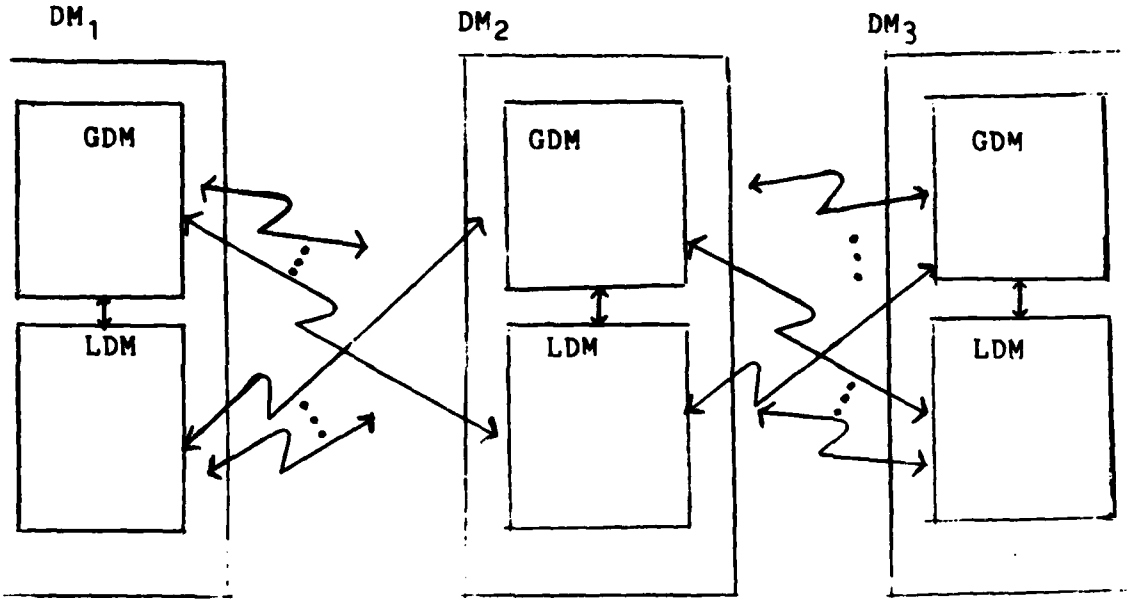
2.2 Datamodule Architecture

This section examines the internal architecture of the SDD-1 datamodules. The datamodule architecture is illustrated in Figure 2.2. As the figure indicates, each datamodule consists of two internal modules called the global data manager (GDM) and the local data manager (LDM) respectively.

The LDM manages data located at the datamodule of which it is a part. It has no awareness of data distribution issues whatsoever. The LDM can apply data management operators to its locally stored data, and it can retrieve and modify locally stored data. The LDM performs these functions in response to requests from GDM's.

GDM's, on the other hand, have no local storage at all and depend entirely on the LDM's for storage resources. Their role is to handle all the data distribution issues in SDD-1. Specifically, the functions of a GDM are the following:

Datamodule Architecture Figure 2.2



- The GDM parses a user request into an internal form which exposes what operations are to be performed and what data is to be operated upon.

- The GDM determines which datamodules house data involved in the requests. This determination involves access to directory information which is managed as we describe in Section 3.

- The GDM decides what operations should be performed by LDM's in the network. The decision process takes two factors into account: one is access ef-

ficiency (see Section 5 and [WONG]) and the other is database consistency (see Section 4, [ROTHNIE et al] and [BERNSTEIN]).

A key motivation for selecting this GDM/LDM structure for the datamodule architecture is the long run goal of employing a variety of existing database management systems in the role of LDM. The LDM operations have been consciously confined to the level of function currently provided by typical database management systems. By equipping GDM's with the rules for constructing their primitive LDM calls in the language of the target LDM, a heterogeneous distributed database system can be achieved.

In the initial SDD-1 implementation all LDM's will be Datacomputers (cf [MARILL and STERN]). Since the Datacomputer was designed and implemented as a database management system without any regard for its future role as an LDM, the feasibility of using the Datacomputer in this way suggests the plausibility of using other DBMS's in this role.

2.3 Data Distribution Concepts

The logical data model supported by SDD-1 is relational. In this section we consider the stored representation of a set of relations in SDD-1.

The assignment of logical data items to the physical storage resources of the datamodules begins with the partitioning of each relation into sub-sets called fragments. Each fragment is defined to be a rectangular sub-set of a relation; i.e. fragments are defined as restrictions and projections of database relations. Furthermore, restrictions are constrained to involve simple predicates as defined by [ESWARAN et al], i.e., boolean conditions of the form:

> Attribute <relational operator> constant
>
> where <relational operator>:= "=","<",">","≤", or "≥".

Also, to avoid updating anomalies similar to those described by [CHAMBERLIN et al] with respect to views, each projection is required to include the primary key.

Figure 2.3 illustrates the partitioning of a PERSONNEL logical relation into fragments.*

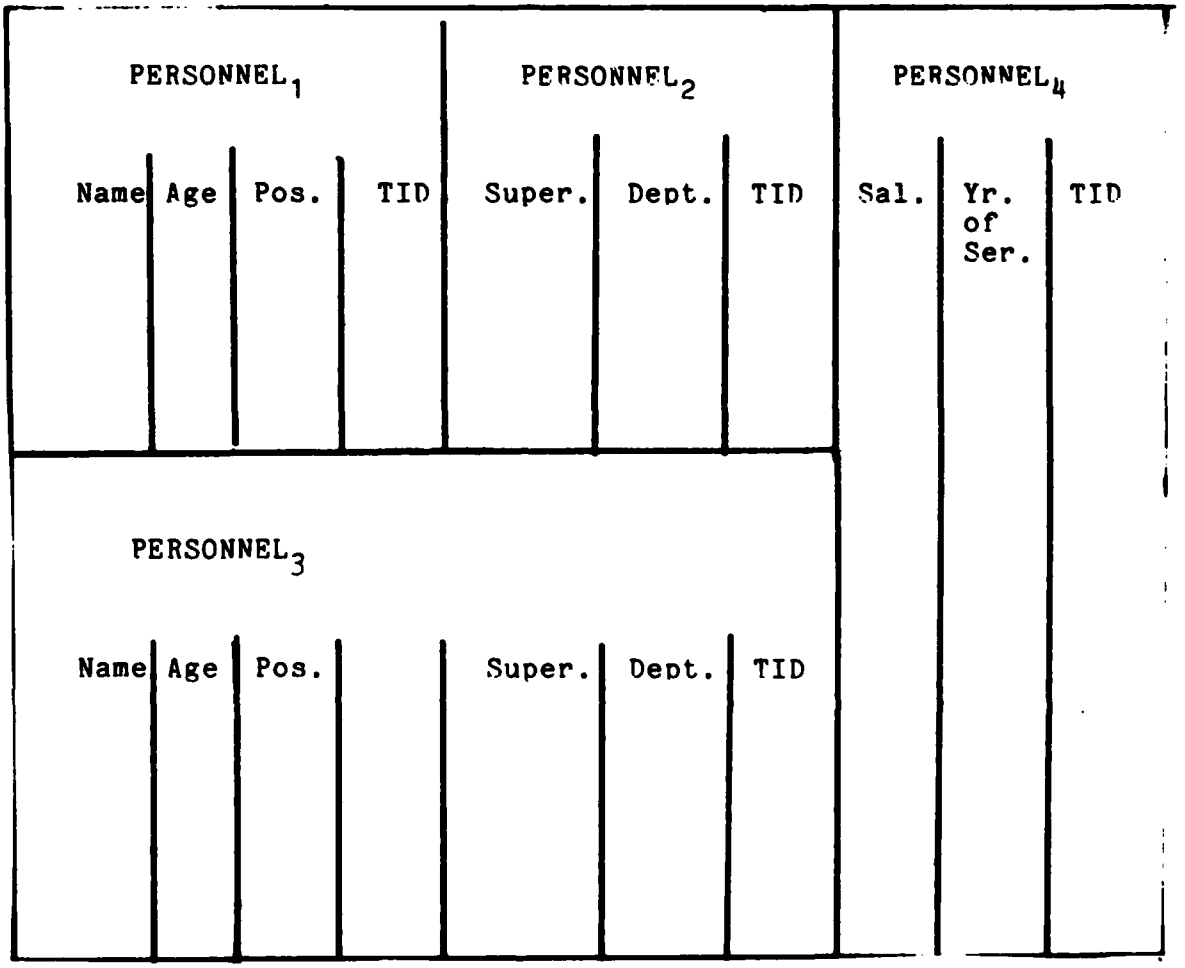
Fragments then, are the units of assignment of logical data items to datamodules. A given fragment is, by definition, either entirely present or entirely absent at each datamodule, DM_i. However, each fragment may be stored redundantly at more than one module. The stored representation of a fragment in a given module is termed a stored fragment and is designated Stored-$F_{i,m}$ meaning the representation of fragment F_i in datamodule m. Figure 2.4 illustrates the partition of a logical database into fragments and the assignment of fragments to modules. Each arc from the rectangles (representing fragments) to the circles (representing datamodules) corresponds to a stored fragment.

Since fragments may be stored redundantly, in general there will be more than one way of reconstructing a complete and non-redundant copy of the logical database from

---

* A way of achieving this requirement without impacting users is to include in every relation an attribute called TID (for tuple ID) which can be used by the system internally but is not visible to users. INGRES [STONEBRAKER et al] and System R [ASTRAHAN et al] use the TID concept for other purposes. Like those uses, though, we assume that TID is guaranteed to be unique for each tuple. In this way, the primary key required in each fragment-defining projection can simply be TID.

Partition of a Relation into Fragments Figure 2.3



Each fragment is defined as follows:

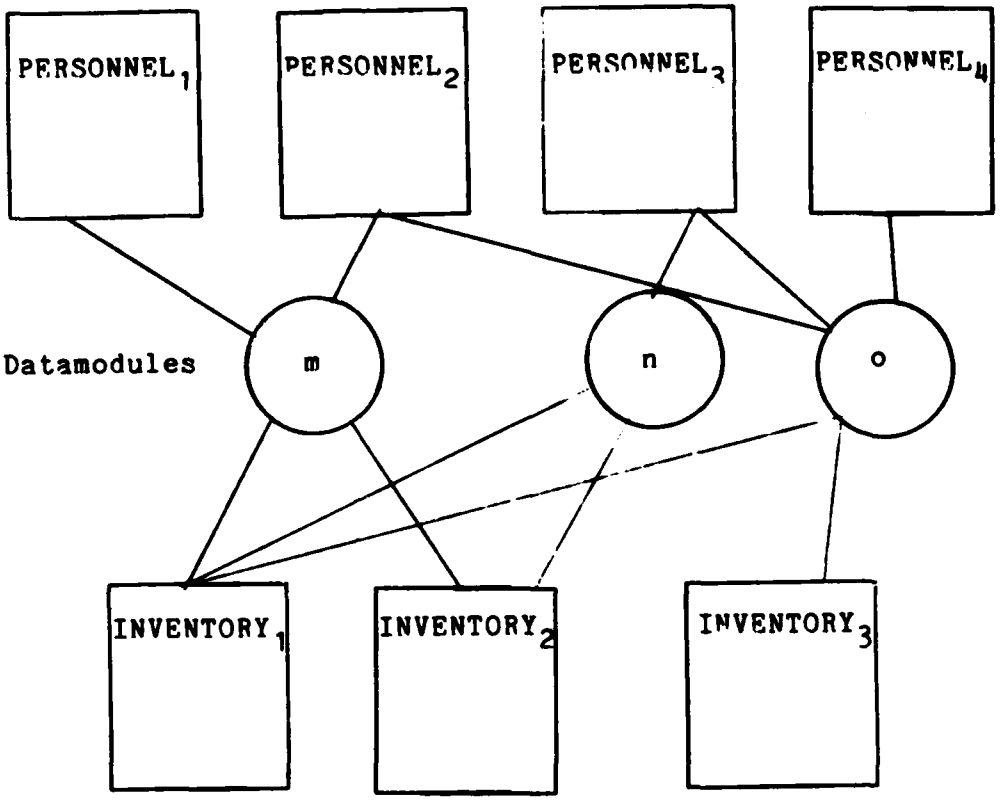
- $PERSONNEL_1$:=
  PERSONNEL where Salary > $30,000,
  projected on Name, Age, Position, TID

- $PERSONNEL_2$:=
  PERSONNEL where Salary > $30,000,
  projected on Supervisor, Department, TID

- $PERSONNEL_3$:=
  PERSONNEL where Salary <= $30,000,
  projected on Name, Age, Position,
  Supervisor, Department, TID

- $PERSONNEL_4$:=
  PERSONNEL projected on Salary,
  Years-of-service, TID

Assignment of Fragments to Datamodules Figure 2.4



the collection of stored fragments. For instance, in the example of Figures 2.3 and 2.4, there are four ways of reconstructing a complete, non-redundant copy of the logical PERSONNEL relation:

1. One which consists of

   Stored-PERSONNEL$_{1,m}$, Stored-PERSONNEL$_{2,m}$, Stored-PERSONNEL$_{3,n}$, and Stored-PERSONNEL$_{4,o}$;

2. Another consisting of

   Stored-PERSONNEL$_{1,m}$, Stored-PERSONNEL$_{2,m}$, Stored-PERSONNEL$_{3,o}$, and Stored-PERSONNEL$_{4,o}$.

3. A third consisting of

   Stored-PERSONNEL$_{1,m}$, Stored-PERSONNEL$_{2,o}$, Stored-PERSONNEL$_{3,n}$, and Stored-PERSONNEL$_{4,o}$; and

4. Finally, one consisting of

   Stored-PERSONNEL$_{1,m}$, Stored-PERSONNEL$_{2,o}$, Stored-PERSONNEL$_{3,o}$, and Stored-PERSONNEL$_{4,o}$.

Similarly there are six ways of reconstructing the INVENTORY relation in that example.

A collection of stored fragments which form a complete and non-redundant copy of the logical database is called a materialization. At any given time SDD-1 recognizes a specific set of materializations as "supported". A user logging in to SDD-1 is given access to the database through a specific supported materialization and repeated accesses to the system will result in assignment to the same supported materialization.

The make-up of each supported materialization is recorded in a table such as the one illustrated in Figure 2.5. The management of this and other directory information will be described in Section 3. Another table indicates the materialization to which each user is assigned. SDD-1 will service a user's retrieval requests by accessing only the stored fragments listed for his assigned materialization.

The concept of supported materializations (hereafter called simply "materializations") is an important element of SDD-1's data distribution model. Since each materialization is a complete and non-redundant copy of the database, it is analogous to the simpler concept of logical relations in the non-distributed database setting. The materialization concept plays a central role in the notion of database consistency in SDD-1 and has a major impact on the problem of updating redundantly stored data. This issue is addressed in Section 4.

Table of Fragment Assignments Figure 2.4

Supported Materialization	PERSONNEL relation fragments 1	2	3	4	INVENTORY relation fragments 1	2	3
1	m	m	n	o	m	m	o
2	m	m	o	o	o	n	o
3	m	o	n	o	m	m	o
4	m	o	n	o	n	n	o
5	m	o	o	o	m	m	o
6	m	o	o	o	n	n	o

Table entry is datamodule from which the indicated materialization obtains each fragment. Note that the table shows supported materializations only and does not show all possible materializations.

3. Directory Management

One issue which is frequently cited (e.g., in [FRY and SIBLEY], [SIBLEY], [STONEBRAKER and NEUHOLD]) as significant in the design of distributed database systems is the management of the global system directory. This directory provides the information which the GDM's require to parse user requests, to locate data of interest, and to choose accessing and updating strategies.

Alternatives for handling the directory include:

- storing it in a single, central datamodule;
- storing a complete copy at each datamodule;
- dispersing the directory over the set of datamodules non-redundantly; and
- dispersing the directory over the set of datamodules with some redundancy.

The optimal choice among these alternatives depends upon the pattern of transaction traffic in the network. Ideally, general purpose distributed database systems would not adopt any of these alternatives directly but rather would permit the choice to be made as part of database design.

SDD-1 achieves this flexibility by treating the directory as ordinary user data. The data which comprise the directory are partitioned into fragments just as all other user data are, and like user data, these fragments are stored in a distributed and redundant manner throughout the system. In this way arbitrary levels of redundancy and distribution of the directory can be supported. It should be noted that treating the directory as user data also makes other system features such as security, integrity, and concurrency control available for directory management.

There is one regard in which directory information cannot be treated identically to user data and that is in the matter of where the directory information for the directory itself is stored. It is necessary to treat this information specially in order to provide the system with a known starting point from which directory information can be found.

This problem is handled in SDD-1 by factoring the directory information for directories into separate relations which are stored in every datamodule. The decision to store this level of directory information at all datamodules is based on the assumption that these relations are small and very retrieval intensive.

4. Efficient Redundant Updating

In developing a distributed database system such as SDD-1, one of the key problems is to update the multiple copies of a single logical data item in such a way that consistency of the database is preserved and excessive synchronization overhead is avoided. The SDD-1 approach to this problem is treated in detail in [ROTHNIE et al] and [BERNSTEIN]. In this section we outline certain key aspects of the technique.

The notion of database consistency is defined for SDD-1 in terms of materializations. Since each (supported) materialization represents a non-redundant logical view of the database as it might be seen by a user, a strong form of consistency must be preserved within each materialization. Indeed the internal consistency of each materialization must be maintained as strongly as the internal consistency of a conventional, centralized database system, i.e:

We first assume that each transaction always maps a consistent database state into another consistent state*. Then, internal consistency can be defined in terms of serializability (cf [ESWARAN et al], [GRAY et al], and [HEWITT]). Serializability dictates that the effect of a set of concurrently executing transactions on a database must be equivalent to the effect of some serial, non-overlapping sequence of those same transactions. Then, since each transaction running separately does not violate database consistency, the effect of the concurrent collection cannot be inconsistent.

This form of consistency is quite strong and it is expensive to preserve in terms of communication costs and processing delays.

Fortunately it is sufficient to maintain a much weaker form of consistency between materializations. All that is required for mutual consistency between materializations is that the database copies they represent not diverge over time. It is not necessary for the copies to be in-

---

* Verifying that separate transactions will not violate consistency is an integrity checking step of the sort described by [HAMMER and MCLEOD]. It will not be considered here.

stantaneously identical at all times; it is sufficient that they would attain the same final state were all update activity to cease.

The simplest method for controlling redundant updates is to lock those portions of the database being read or written by active transactions. However, in a distributed database environment an appreciable and often intolerable delay is introduced while locking information is propagated to the many computers in the database network.

More efficient methods for locking in a distributed database system have been proposed by [THOMAS-b] and by [ALSBERG and DAY]. The method proposed by [THOMAS-b] reduces the volume of inter-module communications needed for locking by utilizing a voting protocol; in this method one need only communicate with a majority of data-modules rather than with all.

The [ALSBERG and DAY] method is quite different and introduces the notion of a "primary site" for updating. In this approach all update activity in the distributed database system must be handled through a single primary update site, and all locking occurs locally within that site.

Although the improvements suggested by these authors may be appropriate in certain contexts, both are infeasible in a general distributed system like SDD-1.

The method used in SDD-1 differs qualitatively from this previous work. Rather than merely trying to improve the efficiency of global database locking the SDD-1 method seeks to avoid global locking whenever possible. While it is true that in general, arbitrary transactions must place locks throughout the database network, there are many cases in which locking need only occur in the datamodule where the transaction was initiated. In these cases, the propagation of the update information to all the other redundant copies of the data is carried out by an efficient protocol which again only requires locking on a module by module basis.

Thus this updating method permits SDD-1 to exhibit the same fast response for most update transactions that it can offer on retrieval. For any transaction the question of whether it may be run without global locking depends on what other transactions are to be run in that manner. This decision is made by the database administrator as part of the database design activity.

The database administrator defines classes of transactions that are commonly executed in the database application and

for each class he specifies which datamodules are expected to enter the transactions. Then by applying the process presented in [ROTHNIE et al] and [BERNSTEIN], the specified configuration of transactions is analyzed to determine the amount of synchronization that is required for transactions in each class.

A table representing the output of the analysis is used by each datamodule at run-time to determine the appropriate synchronization level required by a given transaction. It is important to recognize that this predefinition activity in no way limits the transactions which can be processed by the system. It merely permits more efficient execution of those classes which have been anticipated.

5. Efficient Dispersed Data Access

The distribution of data in systems like SDD-1 represents a potential problem in responding rapidly to complex queries. The execution of a user request involving cross-referencing operations (e.g., joins) over relations at several sites can incur a very substantial delay. This delay is caused by the need to bring together at one processor all the data required for the execution of the cross-referencing operation.

The approach adopted in SDD-1 for handling this problem is described in detail in [WONG]. The approach is similar in a formal sense to the decomposition strategy proposed by [WONG and YOUSSEFI] for optimizing queries in a non-distributed database setting. The algorithm in the distributed environment operates by decomposing a multi-datamodule query into a sequence of local queries involving data at only one datamodule, with inter-module data transfers between the local queries. The execution strategies which are generated by this technique consist of a sequence of steps, each of which is either

a. a set of site-to-site data moves or

b. local processing on data which has been moved to a single site.

The sequence of steps selected is determined by a step-by-step optimization process which can be represented by a binary decision tree.

A key assumption of the optimization technique is that communication costs will dominate the processing of complex queries in a distributed environment. Hence at each step the optimization procedure first attempts to minimize communication costs and then, within the optimal communication strategy, attempts to minimize local processing.

6. Summary

SDD-1 is a distributed database system in the early stages of implementation which is characterized by the following key concepts:

1. It is implemented as a set of communicating processors, called datamodules, which are dispersed worldwide and which are envisioned to number in the hundreds for a single system.

2. The distributed implementation is invisible to users.

3. SDD-1 supports arbitrary redundancy. The physical storage of data items is defined in terms of logical subrelations called fragments.

4. Each datamodule is implemented as two communicating processes, called the global data manager and the local data manager. The partition of function between these two components permits the eventual incorporation of existing database management systems into a heterogeneous SDD-1.

5. Directory information is stored in SDD-1 as ordinary user data. This permits choices regarding the distribution and redundancy of the directory to be delayed until database design.

6. Updating multi-copy data items is a process which can represent an intolerable bottleneck in distributed systems. SDD-1 employs a transaction classification technique which avoids time-consuming inter-module synchronization for most update transactions.

7. The system uses an access planning technique which attempts to minimize communication costs as its primary optimization goal and local processing costs as a secondary goal.

References

[ALSBERG et al]
Alsberg, P. A.; et al. "Synchronization and Deadlock", CAC Document No. 185, Center for Advanced Computation, University of Illinois, Urbana, Illinois, March 1976.

[ALSBERG and DAY]
Alsberg, P. A.; and Day, J. D. "A Principle for Resilient Sharing of Distributed Resources", Report from the Center for Advanced Computation, University of Illinois, Urbana, 1976. (Also accepted for proceedings of the Second International Conference on Software Engineering.)

[ASTRAHAN et al]
Astrahan, M. M.; et al. "System R: Relational Approach to Database Management", ACM Transactions on Database Systems, Vol. 1, No. 2, June 1976, pp. 97-137.

[BERNSTEIN]
Bernstein, P. A. Technical Report in Progress, Computer Corporation of America, Cambridge, Massachusetts.

[CHAMBERLIN et al]
Chamberlin, D. D.; Gray, J. N.; and Traiger, I. L. "Views, Authorization, and Locking in a Relational Database System", Proceedings AFIPS National Computer Conference, AFIPS Press, Vol. 44, 1975.

[ELLIS]
Ellis, C. A. "The Duplicate Database Problem", Request for Comments no. 112, M.I.T. Laboratory for Computer Science, Computer Systems Research Division, Cambridge, Massachusetts, May 11, 1976.

[ESWARAN et al]
Eswaran, K. P.; Gray, J. N.; Lorie, R. A.; Traiger, I. L. "The Notions of Consistency and Predicate Locks in a Database System", CACM, Vol. 19, No. 11, November 1976.

[FRY and SIBLEY]
Fry, J. P.; and Sibley, E. H. "Evolution of Database Management Systems", Computing Surveys, Vol. 8, No. 1, March 1976, pp. 7-42.

[GRAY et al]
Gray, J. N.; Lorie, R. A.; Putzolu, G. R.; and Traiger, I. L. "Granularity of Locks and Degrees of Consistency in a Shared Database", Report from IBM Laboratory, San Jose, California, 1975.

[HAMMER and MCLEOD]
Hammer, M. M.; and McLeod, D. J. "Semantic Integrity in a Relational Database System", Proceedings of the International Conference on Very Large Databases, September, 1975.

[HEWITT]
Hewitt, C. E. "Protection and Synchronization in Actor Systems", Artificial Intelligence Laboratory Working Paper No. 83, Massachusetts Institute of Technology, November 1974.

[MARILL and STERN]
Marill, T.; and Stern, D. H. "The Datacomputer: A Network Data Utility", Proceedings AFIPS National Computer Conference, AFIPS Press, Vol. 44, 1975.

[ROSENTHAL]
Rosenthal, R. "A Review of Network Access Techniques with a Case Study: The Networks Access Machine", NBS Technical Note 917, July 1976.

[ROTHNIE et al]
Rothnie, J. B.; Goodman, N.; Bernstein, P. A.; and Papadimitriou, C. A. "The Redundant Update Methodology of SDD-1: A System for Distributed Databases", Technical Report no. CCA-77-02, Computer Corporation of America, Cambridge, Massachusetts.

[ROTHNIE and GOODMAN]
Rothnie, J. B.; and Goodman, N. "A Study of Updating in a Redundant Distributed Database Environment", Technical Report no. CCA-77-01, Computer Corporation of America, Cambridge, Massachusetts, February 15, 1977.

[SCHANTZ and MILLSTEIN]
Schantz, R. E.; and Millstein, R. E. "The FOREMAN: Providing the Program Execution Environment for the National Software Works", BBN Report No. 3266, March 1976.

[SIBLEY]
Sibley, E. H. "The Development of Database Technology", Computer Surveys, Vol. 8, No. 1, March 1976, pp. 1-5.

[STONEBRAKER et al]
Stonebraker, M.; Wong, E.; Kreps, P.; and Held, G. "The Design and Implementation of INGRES", ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp. 189-222.

[STONEBRAKER and NEUHOLD]
Stonebraker, M.; and Neuhold, E. "A Distributed Database Version of INGRES", Memorandum no. ERL-M612, Electronics Research Laboratory, University of California, Berkeley, September 11, 1976.

[[THOMAS-a]]
Thomas, R. H. "A Resource Sharing Executive for the Arpanet", Proceedings AFIPS National Computer Conference, AFIPS Press, Vol. 42, 1973, pp. 155-163.

[[THOMAS-b]]
Thomas, R. H. "A Solution to the Update Problem for Multiple Copy Databases which Uses Distributed Control", BBN Report No. 3340, July 1975.

[WONG]
Wong, Eugene. "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases", Technical Report no. CCA-77-03, Computer Corporation of America, Cambridge, Massachusetts, March 15, 1977.

[WONG and YOUSSEFI]
Wong, E.; and Youssefi, K. "Decomposition - A Strategy for Query Processing", ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976, pp 223-241.