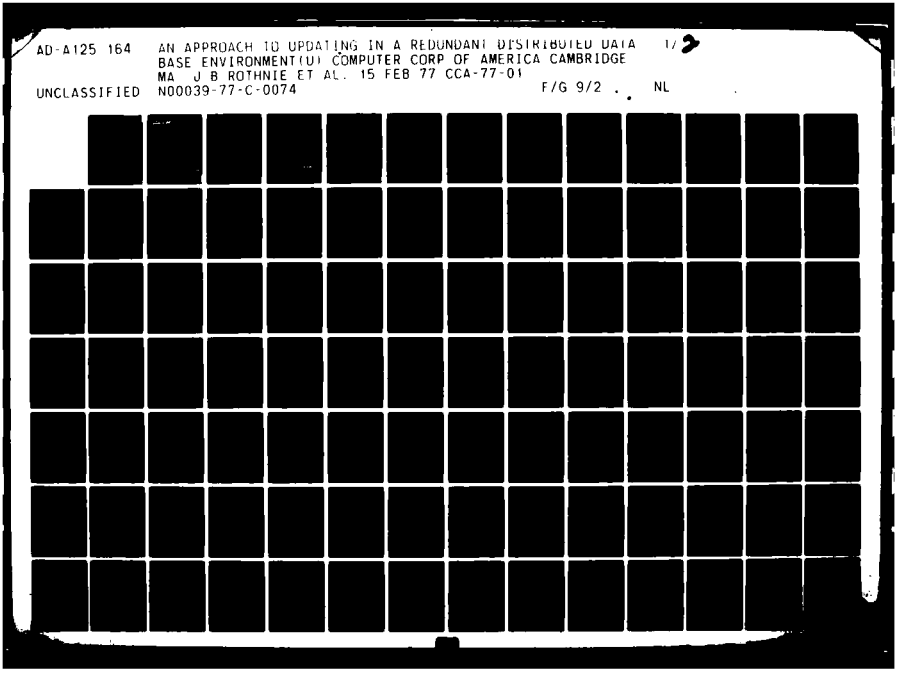
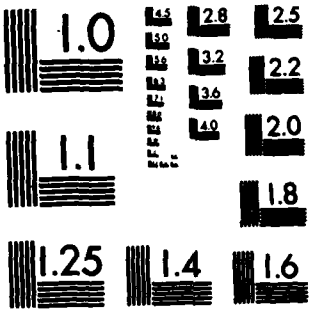

AD-A125 164 AN APPROACH TO UPDATING IN A REDUNDANT DISTRIBUTED DATA 1/2
BASE ENVIRONMENT(U) COMPUTER CORP OF AMERICA CAMBRIDGE
MA J B ROTHNIE ET AL. 15 FEB 77 CCA-77-01
UNCLASSIFIED N00039-77-C-0074 F/G 9/2 NL



An Approach to Updating in a Redundant Distributed Data Base Environment

Technical Report
CCA-77-01
February 15, 1977

J.B. Rothnie
and
N. Goodman

Computer Corporation of America

Computer Corporation of America
575 Technology Square
Cambridge, Massachusetts 02139

A Study of Updating In a Redundant Distributed Database Environment

Technical Report
CCA-77-01

February 15, 1977

James B. Rothnie and Nathan Goodman

Computer Corporation of America

CONTRACT N00039-77-C-0074

This research was supported by the Defense Advanced Research Project Agency of the Department of Defense. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Abstract

This working paper describes an approach to the problem of updating data that is stored in a redundant distributed database system (RDDBS). The approach that we describe:

a. preserves the consistency of the database; yet

b. avoids excessive delays due to inter-computer synchronization.

The technique exhibits substantially faster response to updates than methods suggested elsewhere (e.g. [THOb], [ALS]). In fact the depicted approach offers the same response time advantages for most updates that an RDDBS offers on retrieval. Also, in contrast to the previous work in the field, the method described here is amenable to "tuning" as part of the database design process.

Acknowledgments

The authors wish to acknowledge the contributions of Philip A. Bernstein and Christos Papadimitriou to the results presented in this report.

Updating a Redundant Distributed Database
Table of Contents

1. Introduction

This paper addresses the following problem: how to perform updates in a redundant distributed database system (RDDBS) in a manner that (a) preserves the consistency of the database, yet (b) does not introduce intolerable inter-computer synchronization delays.

We have developed a technique for this problem which permits most update transactions to exhibit the same highly responsive behavior which an RDDBS can offer on retrieval. Other approaches to the problem of updating in a redundant distributed database environment suffer from substantially slower response times (in many cases involving clearly intolerable delays) as well as from problems in system scaling. These approaches are discussed in Section 2.

The results presented in this paper are preliminary in nature. They derive from research conducted by CCA in preparation for a large-scale RDDBS implementation project. This RDDBS implementation is being performed by CCA on behalf of the Defense Advanced Research Projects Agency in the context of Navy command and control applications [CCA]. Through the implemen-

tation effort and additional theoretical research we expect to extend and strength the results presented in this paper.

The body of this paper describes the approach to redundant updating that we have developed. Also it outlines the theoretical questions that we expect to address in future work.

The remainder of Section 1 discusses the general notion of an RDDBS and the major technical problems such a system presents.

Section 2 focuses on the redundant update problem; it presents solutions to the problem which have been prepared elsewhere as well as an overview of the new technique suggested here.

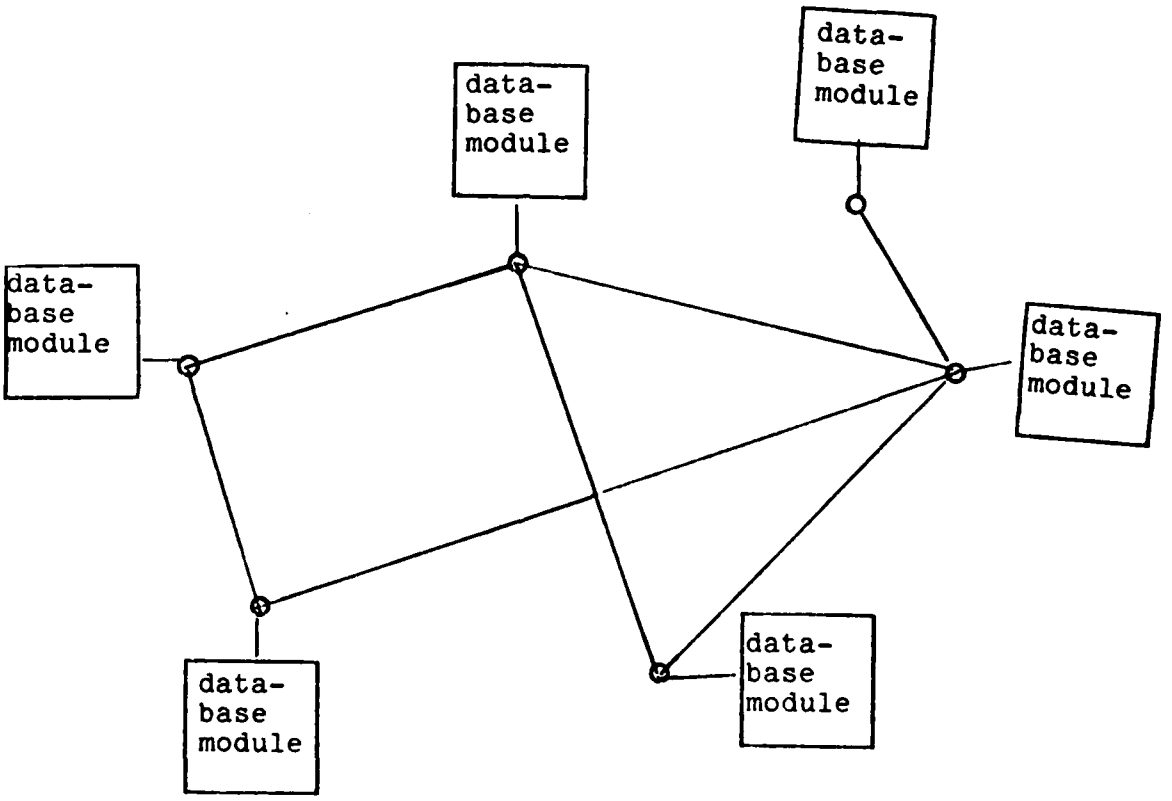
Section 3 provides a more precise formulation of the general system architecture of the RDDBSs being studied.

Sections 4 and 5 describe in detail the redundant update methodology we are advocating. First, Section 4 explores the question of database consistency which underlies the update problem. Then, a graphic technique which simplifies the analysis of potentially inconsistent database operations is introduced. Finally, in Section 5 the graphic technique is exploited to describe and analyze the update methodology and some sample results are given.

1.1 Motivation for RDDBSs

The advent of inexpensive and easy to use computer networks has given rise to interest in the concept of distributed processing. The database world is no exception to this trend in computer technology and over the past several years an increasing amount of research and developement has been conducted on distributed database systems. Early work in this area includes that of Whitney [WHI], Casey [CAS], and Chu [CHU]; more recent work has been conducted by Alsberg [ALS], Thomas [THOb], Stonebraker [STO], Mahmoud [MAH], Levin [LEV], and others.

In the context of this paper, the term "distributed database system" (DDBS) denotes a database management system consisting of several geographically dispersed data base modules connected to each other via a computer-to-computer communication network (see Figure 1.1). The modules are all functionally identical -- that is, no one of them serves any function which could not be done by any other. However, in general each data module serves as a repository for a different portion of the database. Thus in order to satisfy a user's request to store



A Distributed Database System
Figure 1.1

or retrieve data, it may be necessary to access more than one data module.

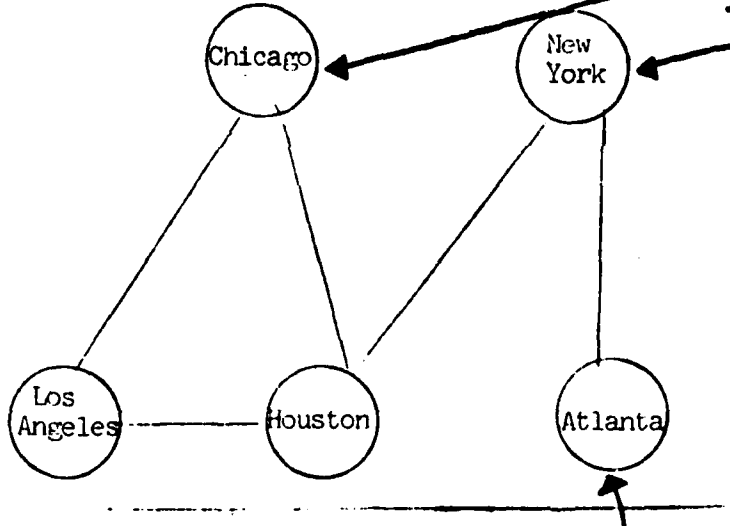
An example of a multi-module query is illustrated in Figure 1.2. The figure visualizes a query issued at corporate headquarters in New York for any employee located at either the company's Chicago or Atlanta office with a special knowledge of compilers and who also speaks German. (This person is perhaps needed for emergency assignment in Hamburg). In the illustrated distributed database system the query requires access to both the Atlanta database module and the Chicago module.

An important nuance in our concept of DDBSs is that this sort of dispersed data access is handled automatically by the system. That is, users are permitted to enter database queries or updates at any database module and the system itself takes responsibility for locating the desired data among the distributed network of database modules. Ideally users are able to interact with the distributed system as easily as with a conventional, centralized one.

Thus the notion of a distributed database system that we are addressing is quite analogous to the distributed operating system concepts being developed today, e.g. by the National Software Works [SCH], the National Bureau of Standards NAM

EMPLOYEE NUMBER	NAME	....	CURRENT STATION	SPECIAL SKILL	FLUENT LANGUAGE	....
123456789	Jones	...	DETROIT	COMPILERS	GERMAN	...
234567890	Brown	...	CHICAGO	NONE	FRENCH	...
345678901	Smith	...	CHICAGO	COMPILERS	GERMAN	...
.	.	.	.	.	.	

This data stored at this module.



For all employees whose
CURRENT-STATION =
CHICAGO or ATLANTA
SPECIAL-SKILL =
COMPILERS,
and
FLUENT-LANGUAGE =
GERMAN
Print EMPLOYEE-NUMBER
and NAME.

This data stored at this module

EMPLOYEE NUMBER	NAME	....	CURRENT STATION	SPECIAL SKILL	FLUENT LANGUAGE	....
456789012	Carter	...	ATLANTA	COMPILERS	SPANISH	...
567890123	Doe	...	ATLANTA	COMPILERS	GERMAN	...
678901234	Gray	...	ATL	COMPILERS	GERMAN	...
.	.	.	.	.	.	.

A Multi-Module Query
Figure 1.2

project [ROS], and the Arpanet RSEXEC project [THOa]. These projects like ours 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 advocated by Thomas [THOb], Stonebraker [STO], and Alsberg [ALS].

Distributed database systems offer advantages over centralized database systems in many applications.

- The database is more reliable in that it is not susceptible to total failure when one piece of equipment breaks down or one geographic location becomes inaccessible.

- By allowing data to be stored near to where it is used most frequently, DDBSs provide faster access to the data and reduce communication costs.

- A third advantage of DDBSs is that they permit modular upwards scaling of database systems: DDBSs permit a very large database to be constructed out of moderate sized database modules instead of a single, very large centralized site; and when more power is needed, the user need only add more database modules in a modular fashion.

The advantages of DDBSs may often be further amplified by permitting redundant data to be stored in them -- i.e. by permitting copies of some portions of the database to be stored at two or more database modules. For instance, it may be advantageous to store information on top executives at both the regional database where they are stationed and at corporate headquarters.

Distributed database systems that permit redundant data are called redundant distributed database systems (RDDBSs). RDDBSs offer the following extra features:

1. Reliability:

   Since multiple copies of data are stored, it is possible for crucial portions of a database to remain accessible even if a database module fails or becomes inaccessible due to communication outages.

   Basically the RDDBS approach recognizes that data per se is a valuable property. As many authors point out (e.g. [SIB], [FRY], and [BER]) the major objective of database management in the first place is to increase the availability of data within an organization. It is worth spending time, effort, and money to increase the availability of the valuable resource represented by data.

2. Responsiveness:

Both redundant and non-redundant distributed systems improve the responsiveness of data access by permitting data to be stored near to where it is used. However, in the non-redundant case, the decision on where to place a datum is an all-or-nothing choice, whereas in an RDDBS the decision is subject to "tuning" by a database administrator.

For example, if some data were accessed equally from Los Angeles and from New York there might be no good choice for its location in a non-redundant system. In a redundant DDBS, though, the database administrator could freely choose to store the data both places in order to optimize database performance.

3. Upwards scaling:

The redundant approach allows additional database modules to be added to accommodate increases in database <u>activity</u>, not just increases in database size. As more and more database requests are directed at an area of a database, it could be redundantly stored at additional database modules thus bringing added computing power to bear in handling the activity.

As a consequence of these features RDDBS technology appears appropriate for many Defense database applications. One area where the RDDBS approach is extremely beneficial is the command and control field. Here the requirement for reliable, continuous operation is paramount and this need cannot be met under battlefield conditions by simply placing redundant hardware in a single central location. Also the need for rapid response without excessive communication is clear in the command and control environment.

1.2 Overview of Technical Problems in RDDBSs

As indicated above, RDDBS technology shows great potential for improving the reliability and responsiveness of many database applications. Before this potential may be achieved, however, several technical problems remain to be solved.

1.2.1 The Redundant Update Problem

This paper focuses on one large area of difficulty in the RDDBS field: the problem of updating data that is stored redundantly at multiple database modules. This so-called redundant update problem is in a sense the flip side of the key RDDBS advantages.

> RDDBSs are advantageous on retrieval because they maintain up-to-date data at many sites; but in order to keep its copies up-to-date, an RDDBS must ensure that modifications take affect in all sites in a timely fashion.

Solutions to the redundant update problem have appeared in the literature (e.g. [THOb], [ALS]). These solutions, however, impose a significant and often intolerable delay on the process of updating the database. These solutions thus remove one of the key features of RDDBSs -- their responsiveness -- from the update operation. The previous work on the redundant update problem is discussed further in Sections 2.3 and 2.4.

The approach that CCA will follow in redundant updating permits most update transactions to run as rapidly in the redundant distributed environment as in a non-redundant system. The approach that we pursue has two main foci:

1. We study in precise sense what is meant by "database consistency" and under what conditions transactions in an RDDBS can violate the property of "consistency".

2. We search for local criteria by which to judge each update transaction in order to determine whether it can conceivably violate database consistency. "Local criteria" are ones that can be evaluated at a single database module, without requiring interaction with other database modules.

   If based on these local criteria a transaction is shown to be safe, meaning that it cannot conceivably cause a violation of database consistency, then the update can be propagated throughout the RDDBS using a simple and efficient protocol called the safe protocol.

   Also, from the point of view of the user, safe transactions may be considered as completed as soon as

they are entered into the user's local database module. In other words, the user need not be delayed while the update is propagated to the rest of the RDDBS.

With the CCA technique, therefore, RDDBSs may react responsively to update operations, just as they do to retrievals.

Only when a transaction that is not safe is entered into the system, must the user be delayed while the update is propagated throughout the RDDBS. A major thrust of this research is the discovery of safe transaction classes that may be used to advantage in specific applications; this issue is taken up in Section 5. As discussed there, several large classes of such transactions are already identified.

The remainder of this document, starting with Section 2, is devoted to the redundant update problem. It is important to keep in mind, though, that other significant problems remain to be solved in constructing practical RDDBSs. These issues must be recognized in order to avoid the danger of designing a redundant update solution which does not fit optimally in the larger context of a complete RDDBS.

The rest of Section 1 outlines the outstanding problems in the RDDBS area other than the redundant update problem.

1.2.2 Resiliency Despite Component Failure

In order for RDDBS to be practical they must be capable of withstanding the failure of individual components. Following Alsberg [ALS] we term this property "resiliency".

The general problem is to handle failures in such a way that user service and database consistency is not threatened. This can be partitioned into the following sub-problems:

a. Recognizing a failure
   All active database modules must detect the failure of another database module so that they can formulate responses to user requests without attempting to access the failed module.

b. Recovering a failed database module
   When a failed database module is again ready to provide service, it is necessary to inform the rest of the database modules that it is available again

and to bring its local data up-to-date with the system-wide database.

c. Recovering transactions affected by the failure

If a database module fails during the processing of a transaction, a mechanism is necessary to continue processing without the cooperation of the failed database module. Also, the system must ensure that the partial results obtained while the failed database module was up do not cause a database inconsistency.

d. Recovering from a partitioned network

The most difficult recovery problem in a database module network is resynchronizing a set of database modules which have been partitioned because of communication failure. The problem stems from the possibility of mutually inconsistent databases existing in the isolated data parts.

1.2.3 Optimal Resource Allocation

Many important issues remain to be investigated concerning the optimal allocation of database module and communication channel capacities in an RDDBSs. Among these questions are

- where to locate database modules in a large network;

- which data should be stored redundantly, how many copies of it should be stored, and where should it be stored;

- the bandwidth of the links in the communication network that inter-connects the database modules, and in the network that connects the RDDBS to application hosts.

This area has been studied extensively (e.g by Whitney [WHI], Chu [CHU], Casey [CAS], and Levin [LEV]) for non-redundant distributed database systems. Recently Mahmoud [MAH] has investigated some of these questions for the redundant case, although in a simplified setting. Much work in this area remains to be done.

Our redundant update solution is geared toward database design or tuning and hence is compatible with optimal resource allocation concepts. For instance, our approach allows any data to be stored redundantly but does not require that all portions of the database be so stored.

Further interesting optimality questions arise out of our treatment. These concern the effect of the redundant update algorithm itself on resource allocation. For instance if some datum will usually be updated via the locking protocol, then it tends to be better to store it non-redundantly; this is because the fewer the modules where it is stored the less synchronization overhead there is in updating it. However, if the datum will usually be updated through the safe protocol, then it may be better to store it more redundantly in order to improve the responsiveness of the system.

1.2.4 Data Access Optimization

Transactions in an RDDBS (retrievals or updates) may in general reference data stored at multiple database modules. A number of issues arise concerning how to optimize references to this dispersed data so as, for instance, to minimize the amount of data transferred between database modules in processing the transaction.

Consider the database illustrated in Figure 1.3. The PEOPLE file contains NAMEs and AGEs of people but does not include the type of CAR owned by the person. The CAR file contains that information. The following transaction cross-references the two files to print the names and ages of the people who own red Fords:

```
T :  For any person in the PEOPLE file, let PERSON-NAME: = NAME
                                       and let PERSON-AGE: = AGE
     For any car in the CAR file with OWNER = PERSON-NAME
                                  and   MAKE = 'FORD'
                                  and  COLOR = 'RED'

          Print PERSON-NAME, PERSON-AGE
```

PEOPLE

	NAME	AGE	SEX	...
A person →	Smith	23	M	...
	.	.	.	.
	.	.	.	.
	.	.	.	.

CARS

	OWNER	MAKE	COLOR	YEAR	...
A car →	Smith	FORD	RED	1911	...
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.

Example of Dispersed Data Access
Figure 1.3

```
T: For any person in the PEOPLE file,
       let PERSON-NAME:=NAME

   and let PERSON-AGE:=AGE
       For any car in the CAR file
            with OWNER=PERSON-NAME
            and MAKE='FORD'
            and COLOR='RED'

            Print PERSON-NAME, PERSON-AGE
```

Suppose that the PEOPLE file were stored on one database module, call it DM-PEOPLE, and that the CARS file were stored on another module, DM-CARS. Assume also that the transaction were initiated at the DM-PEOPLE module.

A straightforward way of executing the transaction would have the entire CARS file transmitted to DM-PEOPLE each time through the outer loop. If there were n records in the PEOPLE file, then the entire CARS file would be transmitted n times to the receiving database module. Not only would the cost of processing the request be affected, but so would the speed. Compared with local disk transfer times of 6 megabits per second, the ARPA network is at least 120 times slower (assuming the maximum network bandwidth of 50 kilobits per second).

In order to improve this performance, a number of optimizations are possible:

1. Prior to entering the outer loop, request that DM-CARS transmit to DM-PEOPLE all of those records from the CARS file which have COLOR = RED and MAKE = FORD. This greatly reduces communication relative to the basic looping approach but it requires storage for the transmitted CARS records in DM-PEOPLE. If the transmitted set is too large, this alternative is unsuitable.

2. Enter the outer loop and select a PEOPLE record. Then, instead of asking DM-CARS to transmit the complete CARS file, simply request records which are RED FORDS and which have OWNER = the NAME of the selected PEOPLE record. This will further reduce communications volume relative to improvement 1, if there are some RED FORDS whose OWNER's are not in the selected PEOPLE file.

   In this example, each CAR record would be transmitted only once since each car has only one owner. However, for queries to files not having this characteristic, the use of this improvement will result in multiple transmissions of the same record. Improvement 3 addresses this problem.

3. Process as in 2, except maintain in DM-PEOPLE a buffer containing the last several CAR records transmitted. DM-CARS module keeps track of the records it has transmitted and never retransmits a record which is currently in the buffer. The size of the buffer can be adjusted to reflect the relative costs of storage vs. communication.

The above methods address only the low level communication strategy used to process a transaction. Other, more high level optimizations are possible as well.

One type of high level optimizations involves saving the results of the evaluation of the various booleans and feeding the results of the partial evaluation back into the remainder of the request execution. These operations, called query feedback, are discussed by Rothnie [ROT].

The following example illustrates query feedback. Assume we have a file of ships, containing on board inventories of various items, and another file which has each possible item associated with its "importance". Importance identifies certain items as critical and gives the required amount that must be maintained on board. We wish to build a list of all those ships which are short of some critical item and schedule a call to a supply port. Conceptually, this problem is solved

by two nested loops over the ships file and the item file. The first choice that must be made is whether to consider the item file or the ship file as the "driving loop". For each record in the driving loop, we will examine all the records in the other file. Assume we have chosen the item file as the driver. Naively, then, we would examine every ship once for each item. If there were n ships, and m items, this would require nm references to the ship file. However, since the purpose of the request is to build up a list of ships, once we have determined that a particular ship, say the John F. Kennedy, is short of any item, we need not examine it again. To do so would be redundant in this case since if a ship is short of any critical item it is not necessary to check all the other items as well.

An important characteristic of the above options is that it is not possible to choose a single strategy which is always optimal. A strategy which is good in one case can be bad in another. What must be found are criteria which enable the system to pick the best strategy on a case by case basis.

1.2.5 Storing of the Database Directory

Some attention has been directed in the literature to the question of how the database directory is maintained (e.g. [FRY], [SIB], [STO]). For instance, is the directory stored in one central location, is the entire directory stored at each database module, or is there some other, better method?

Our treatment employs a more flexible method. We view the directory as a normal data file. Therefore it may be stored in one place, it may be stored redundantly in many database modules, or it may be stored partially at many different sites. As we will describe later, our technique for storing redundant data permits small subsets of a file to be stored at a database module, without requiring that the entire file be stored there. Consequently, it is possible to store at each database module just those portions of the database directory that are frequently accessed from that module.

2. The Redundant Update Problem in Detail

In this section we explore the redundant problem in more detail. Two aspects of the problem may be identified:

1. It is necessary to propagate updates from the database module where they are initiated to all other database modules that contain redundant copies of the modified data. This aspect of the problem is not difficult at today's state-of-the-art: it requires the presence of a communications channel such as the Arpanet and a suitable communications protocol.

2. The difficult aspect of the redundant update problem is ensuring that update activity does not violate database consistency. And the key component here is ensuring that concurrent update transactions do not interfere with each other in ways lead to inconsistent database operation.

   Preserving database consistency is of concern to centralized database systems as well as to RDDBSs and the problem has been studied extensively in that

setting (for example by [CHAa,b], [GRA], [ESW]). The classical solution to the problem in the centralized case is to lock data that is accessed by a transaction. The locking approach may be extended to the RDDBS case, but that solution entails a large quantity of communication between database modules in order to achieve the required synchronization.

The focus of the redundant update problem, therefore, is to provide a solution that is better (more efficient) than the straightforward locking solution.

2.1 Example of Updates that Conflict

Consider the RDDBS illustrated in Figure 2.1. The figure depicts an RDDBS containing a personnel file for a nationwide corporation. For the purposes here let's focus our attention on the data for one employee, employee SMITH. SMITH's tuple in the database is as follows:

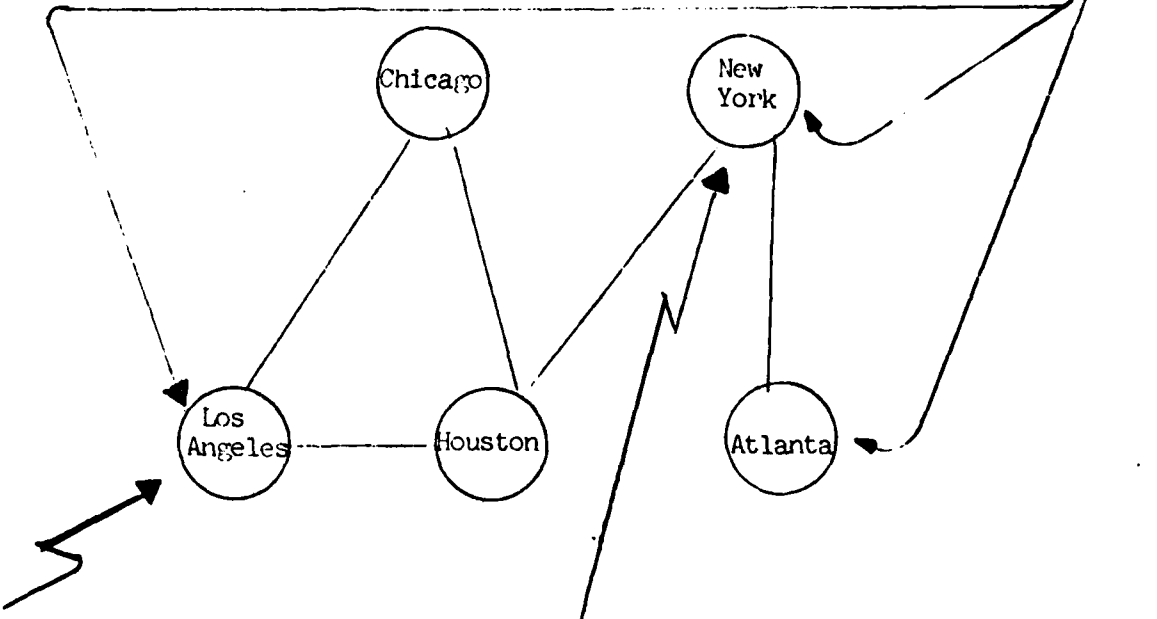
EMPLOYEE-NUMBER	NAME	POSITION	GRADE	SALARY ...
123456789	Smith	Receptionist	IV	$8,000

As Figure 2.1 indicates this tuple is stored redundantly in the New York, Atlanta, and Los Angeles database modules.

Consider also two update transactions that are applied to the RDDBS at approximately the same time:

EMPLOYEE NUMBER	NAME	POSITION	GRADE	SALARY	...
123456789	SMITH	RECEPTIONIST	IV	$8,000	

This tuple stored at these database modules



T2: For all employees whose
POSITION-RECEPTIONIST
and GRADE = IV
Update GRADE to V
and SALARY to $9,000

T1: For all employees whose
EMPLOYEE-NUMBER
= 123456789
Update POSITION to SECRETARY
GRADE to III
and SALARY to $10,000

Personnel File in a Nation-wide RDDBS
Figure 2.1

1. T1, initiated at New York:
   T1: For the employee
   whose EMPLOYEE-NUMBER is 123456789

   Update POSITION to Secretary,
   GRADE to III, and
   SALARY to $10,000.

   The intent of T1 is to promote SMITH from a Receptionist, GRADE IV with a salary of $8,000, to a Secretary, GRADE III receiving a $10,000 salary.

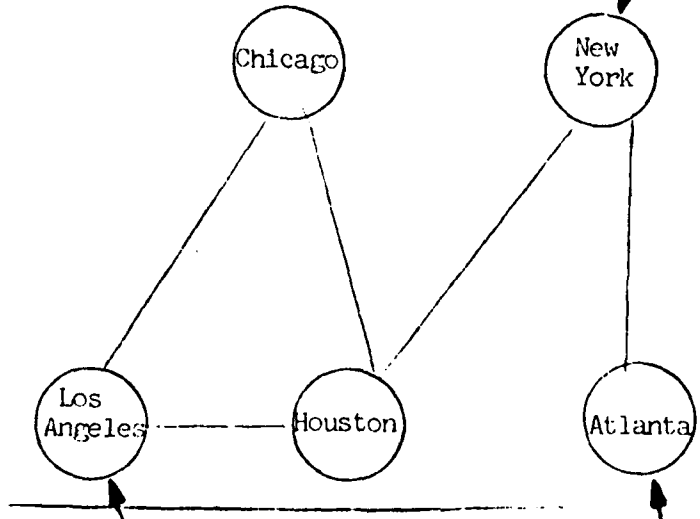
2. T2, initiated at Los Angeles:
   T2: For all employees
   whose POSITION=Receptionist and
   whose GRADE=IV,

   Update GRADE to V, and
   SALARY to $9,000.

   The intent of T2 is to upgrade all GRADE IV Receptionists to GRADE V with a commensurate increase in salary. (Possibly T2 is executed by the personnel department as an automatic "step-raise" procedure).

Figure 2.2 illustrates the state of the database immediately after T1 and T2 are incorporated into their local database modules, but prior to their propagation through the distributed system. Clearly it is obligatory to define a method for propagating the effects of T1 and T2 throughout the RDDBS; otherwise the multiple copies of the database would diverge

EMPLOYEE NUMBER	NAME	POSITION	GRADE	SALARY
123456789	Smith	SECRETARY	III	$10,000



EMPLOYEE NUMBER	NAME	POSITION	GRADE	SALARY
123456789	Smith	Receptionist	V	$9,000

EMPLOYEE NUMBER	NAME	POSITION	GRADE	SALARY
123456789	Smith	RECEPTIONIST	IV	$8,000

Database After T1 and T2 are Incorporated Locally
Figure 2.2

over time and would rapidly cease to be identical copies of each other.

What this propagation protocol must do in essence is merge together the effects of T1 and T2 so as to come up with the total effect of the two transactions on the database. Furthermore, the propagation procedure must be deterministic in the sense that the result of the merge will be identical in all database modules (otherwise the copies would diverge). The operation of this propagation activity in and of itself is not difficult and we present a number of methods for performing it shortly.

Even assuming that a suitable propagation technique is established, however, the example here still poses certain difficulties. Attempts to merge T1 and T2 may result in anomalous database behavior because in a certain sense T1 and T2 contradict each other: T1 is trying to promote SMITH to a Secretary, GRADE III while T2 is trying to promote him to a Receptionist, GRADE V.

There are at least two results possible from merging T1 and T2:

	EMPLOYEE-NUMBER	NAME	POSITION	GRADE	SALARY	...
#1.	123456789	Smith	Secretary	III	$10,000	
	(results if T1 overwrites T2)					
#2.	123456789	Smith	Secretary	V	$9,000	
	(results if T2 overwrites T1)					

Result #2 is particularly bad because it claims that SMITH is a SECRETARY, GRADE V although no one intended he be promoted that far. Furthermore, the SALARY stated for SMITH is no doubt too low for that POSITION. What has happened is that the database has lost its internal consistency.

2.2 Locking Solution to the Redundant Update Problem

The preservation of internal database consistency is an issue in centralized database systems as well as in RDDBSs and many solutions to the problem are available for that environment. Generally, the solution to the consistency problem in centralized database systems may be characterized as follows:

1. Each individual update transaction is pre-checked and subjected to validation tests before it is accepted by the database system. Only transactions that individually do not violate database integrity are allowed to be executed.

2. After a transaction is deemed acceptable by itself, the database system determines whether or not the transaction conflicts with any other concurrent transactions in the database. Conflict detection normally utilizes a locking mechanism, wherein the transaction attempts to place a sharable lock on data it wishes to read and it attempts to place an exclusive lock on data it intends to modify. In some

systems the locks are binary variables, in other systems the locking may employ more complex constructs such as semaphores [DIJ] or monitors [HOA].

If the transaction succeeds in placing all its locks it is guaranteed not to be in conflict with any concurrent transaction and it may proceed to execute. By the same token the transaction is assured that no future transaction that conflicts with it will be allowed to run until the earlier transaction releases its locks.

3. When the transaction is completed it releases all the locks it has set, and other transactions that may have been blocked by it may now enter the database.

Referring back to the example in Section 2.1, it seems plausible that T1 and T2 would each have passed step (1); each of them appears to be okay individually. However, the transactions conflict with each other: T2 reads SMITH's POSITION while simultaneously T1 is writing that datum; also both transactions write SMITH's GRADE and SALARY. In the classical locking protocol for a centralized database, one or the other transaction would have entered the system first and set its locks; the other update transaction would have been blocked

by these locks and prevented from running until the first transaction completed.

A straightforward extension of the locking protocol to the distributed environment is outlined in Figure 2.3. This locking protocol operates inefficiently in two respects:

1. The locking protocol requires many inter-computer messages to be sent in performing a transaction. Let T be a transaction, and let n= the number of database modules that contain data affected by T. The number of messages that must be sent between database modules in order to execute T is

```
    n - 1  lock requests
+   n - 1  lock acknowledges
+   n - 1  updates
+   n - 1  update acknowledges
+   n - 1  lock releases
-----------
   5n - 5  network messages
```

2. The delay perceived by the application process is lengthy. It is:

   the maximum delay encountered in setting the locks
   +
   the maximum delay encountered in performing the update

   In other words, when performing an update transaction by the locking protocol, the RDDBS appears maximally far from all application hosts. The locking solution

LOCKING PROTOCOL

1. TRANSMIT LOCKS TO ALL DATA MODULES.

2. WAIT FOR ALL ACKNOWLEDGEMENTS.

3. TRANSMIT UPDATES TO ALL DATA MODULES.

4. WAIT FOR ALL ACKNOWLEDGEMENTS.

5. TRANSMIT LOCK RELEASES TO ALL DATA MODULES.

6. TRANSMIT ACKNOWLEDGEMENT TO APPLICATION PROCESS.

The Locking Protocol
Figure 2.3

to the redundant update problem thus nullifies one of the key features of RDDBSs -- their responsiveness -- insofar as update transactions are concerned.

For this reason the classical approach to transaction conflict detection cannot be applied effectively in RDDBSs, and other techniques are called for.

Two variations on the classical locking approach to this problem have appeared recently in the literature. Each of these solutions has drawbacks, however, which we hope to correct in our method. These two solutions are examined in Sections 2.3 and 2.4 below.

2.3 Voting Solution to the Redundant Update Problem

Thomas [THOb] has described a solution that cuts down on the total amount of communication required to achieve the locking. This solution requires that only a majority of the database modules approve the lock setting, rather than requiring that all modules approve. Also, this method piggybacks the "lock request" step onto the "transmit update" step thereby eliminating one "transmit message" - "receive acknowledge" sequence from the protocol. * Thus, the voting solution improves on the classical locking protocol in terms of total intercomputer communication required.

However, this method is worse than the classical solution in terms of perceived delay. In the classical locking scheme the messages sent from the initiation site to the other database modules are broadcast in parallel; in the voting method a sequential daisy chain of communication is required. Furthermore, even if a broadcast version of this method could be designed, a significant number of processors -- (n/2 + 1) --

* However, this piggybacking reduces network traffic only in cases where the "transmit update" messages are relatively short themselves, and/or there are few lock conflicts.

must respond to each update. We believe that the voting scheme will perform unacceptably for many update requirements using either the designed daisy chain approach or a broadcast version.

2.4 Primary Site Solution to the Redundant Update Problem

A rather different approach has been proposed by Alsberg [ALS]. This approach requires that all update activity in the RDDBS be funneled through a single database module called the primary site. All locking operations are performed within that single database module and therefore no additional synchronization communication is required.

Alsberg's approach thus avoids the excessive synchronization of the locking protocol but it introduces a number of drawbacks of its own:

a. The responsiveness of the RDDBS to update transactions is no better than the responsiveness of a centralized database system located at the primary site. In a large network, some application hosts will tend to be far away from the primary site and for these hosts system response is likely to be slow.

b. There is no capability for modular upwards scaling of the system. If the amount of update activity exceeds the computational capacity of the primary site, the only recourse is to get a bigger machine. For reliability reasons, the primary site method requires that all database modules be capable of serving as the primary site, and therefore, one must upgrade all database modules in order to add any increased capacity for updating.

c. The notion of having a single site for updating appears to be a highly inappropriate model in the world-wide command and control environment that we are addressing. Imagine a command and control database, portions of which reside on Navy vessels throughout the world. Where would the update site for such a database be placed?

The primary site solution may be appropriate in certain limited applications. But for general use a more flexible approach is necessary.

2.5 Overview of CCA Solution to the Redundant Update Problem

Neither the Voting method nor the Primary Site method completely solves the redundant update problem in a satisfactory manner. We classify both methods as variations of the classical locking protocol in that their fundamental basis involves locking the database for all transactions. Other variations of the locking protocol are possible, of course. We conjecture, though, that no method whose fundamental basis is global locking will fare much better than these two approaches.

The approach that CCA intends to pursue is qualitatively different from the methods described so far; our central paradigm is to avoid global locking by identifying cases where locking is not required.

The approach that we contemplate is based on the observation that many update transactions in the real world may be run concurrently without conflict and without possibility of destructive interference among them. If such transactions are run via a locking protocol, the effort spent in setting the locks will always be wasted since the transactions will never actually conflict. These transactions could instead have been

run using a quicker and more efficient protocol that does no global locking.

Consider the RDDBS and the set of transactions illustrated in Figure 2.4. Each of the transactions in Figure 2.4 is updating the POSITION, GRADE, and SALARY of a specific employee. It seems intuitively clear that these four transactions could be run concurrently without global locking. Of course, not all transactions may be handled this way; later sections of this paper (sections 3,4, and 5) are concerned with characterizing those transactions that don't need global locking vs. those that do. For now, the point is that transactions that don't require global locking can be handled via an efficient protocol called the safe protocol.

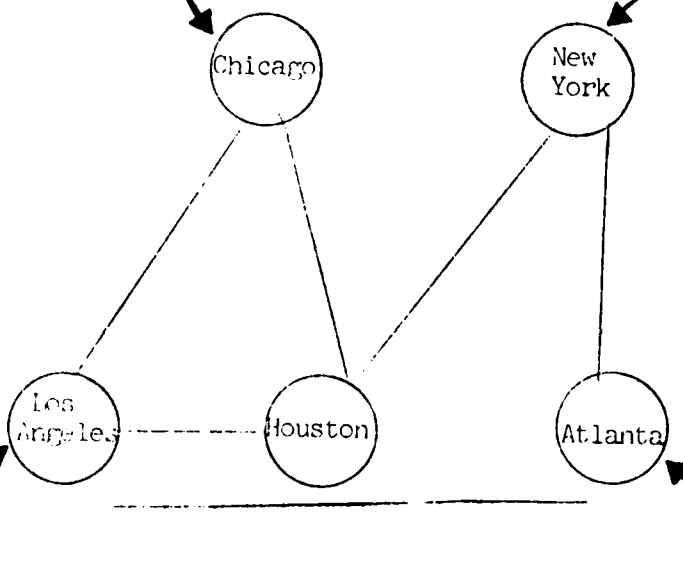
The safe protocol is outlined in Figure 2.5.

Comparing the safe protocol to the locking protocol (see Figure 2.3) we identify two major advantages:

1. The safe protocol requires many fewer network messages. The locking protocol requires approximately 5n messages to propagate an update to n database modules; the safe protocol needs only n messages.

For the employee whose
EMPLOYEE-NUMBER = 456789012
Update POSITION to MANAGER,
GRADE to VI ,
and SALARY to $22,500

For the employee whose
EMPLOYEE-NUMBER = 123456789
Update POSITION to SECRETARY,
GRADE to III,
and SALARY to $10,000



For the employee whose
EMPLOYEE-NUMBER = 345678901
Update POSITION to ANALYST,
GRADE to II,
and SALARY to $18,000,

For the employee whose
EMPLOYEE-NUMBER = 234567890
Update POSITION to SALESMAN,
GRADE to VIII,
and SALARY to $5,000

Four Safe Transactions
Figure 2.4

SAFE PROTOCOL

1. RUN TRANSACTION IN INITIAL DATA MODULE AND PERFORM UPDATES THERE.

2. WHILE RUNNING TRANSACTION, LOG ALL RECORD CHANGES AND TIME STAMP WITH TIME OF TRANSACTION.

3. TRANSMIT ACKNOWLEDGEMENT TO APPLICATION PROCESS.

4. TRANSMIT RECORD CHANGE LOG TO ALL DATA MODULES.

5. RECEIVING DATA MODULES MAKE CHANGES IF TIME STAMP OF TRANSACTION IS MORE RECENT THAN TIME STAMP OF ITEM BEING CHANGED IN DATABASE.

The Safe Protocol
Figure 2.5

2. The delay experienced by the application host is much shorter with the safe protocol. This is because the database module where the transaction was initiated is able to acknowledge the completion of the transaction before it propagates the transaction to any other database modules.

   In other words, when performing an update transaction by the safe protocol, the RDDBS appears maximally close to all application hosts. The safe protocol thus carries forward one of the key features of RDDBS -- their responsiveness--and makes that feature apply to updates just as it applies to retrievals.

Our solution to the redundant update problem is built around the safe protocol. The database system subjects all transactions to tests that determine if the transaction is safe. Safe transactions are run via the efficient safe protocol. Only transactions that are not safe must encounter the inefficiency and the delay of the locking protocol.

The goal of our research is to discover characterizations of transaction classes that permit commonly occurring applications to utilize the safe protocol frequently and thus run efficiently. We do not expect this research to result in a uni-

versally optimal partitioning of transactions into safe vs. not-safe classes. Instead, we expect this partitioning to be a part of the database design process, to be performed, analyzed, and improved by a database administrator with knowledge of the database application.

3. General System Architecture of RDDBSs

This section provides a more precise description of the general system architecture of the data module network. This description provides the context necessary for understanding the redundant update mechanisms discussed in Sections 4 and 5. It should be noted that the architecture discussed here is motivated both by the requirements of the redundant update problem and by other aspects of distributed database technology (such as needs for retrieval efficiency and robustness) which are not the main focus of this paper.

3.1 The Network of Data Modules

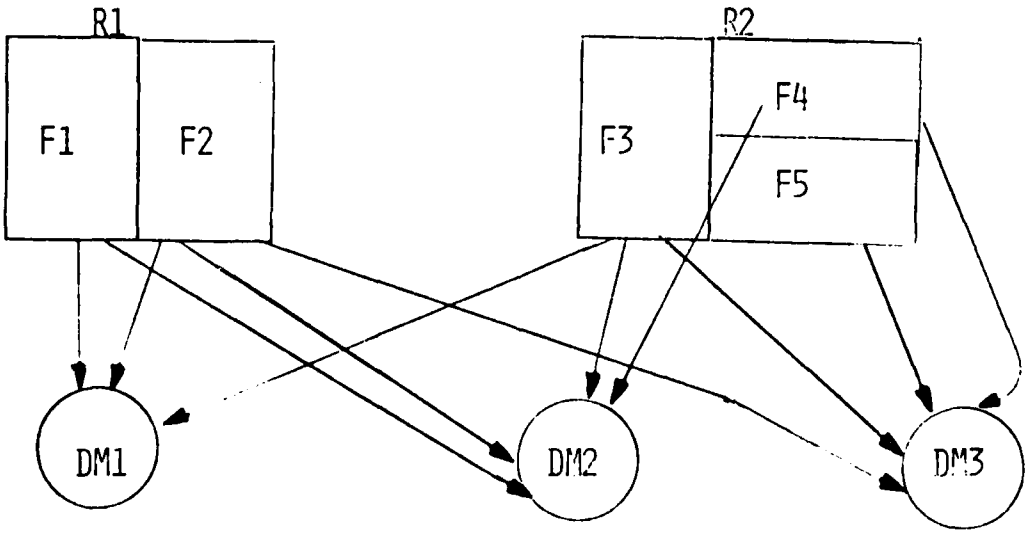
The redundant distributed database system (RDDBS) we are addressing consists of a set of data modules which communicate among themselves and with application hosts over a resource sharing network such as the Arpanet. A member of the set of data modules will be designated DMi and an application host will be referred to as AHi.

The RDDBS presents each AHi with a view of a single non-redundant and non-distributed database. That is, when AHi connects to given data module, DMi, that module will communicate with the rest of the net, as necessary to create the illusion that DMi has a non-redundant copy of the complete database locally resident. For concreteness and simplicity we will stipulate that the database is relational and in third normal form (3NF) [COD]. No important conclusions in this discussion depend on that assumption, however, the presentation is simplified by use of this constrained data model.

3.2 Physical Database Organizations

The assignment of logical data items to the physical storage resources of the data modules begins with the partitioning of the logical database into subsets called fragments (Fi). Each Fi is a rectangular subset of a relation.

The fragment is the unit of assignment of logical data to data modules. A given fragment is either entirely present or entirely absent from each DMi. A fragment may be stored at more than one module. The stored representation of a fragment in a given module is termed a stored fragment and is designated SFi,j to mean the representation of Fi in DMj. Figure 3.1 illustrates the partition of the logical database into fragments and their assignment to modules. Each arc from the rectangles representing fragments to the circles representing data modules corresponds to a stored fragment.



ASSIGNMENT OF FRAGMENTS TO DATA MODULES

Figure 3.1

3.3 Materializations

Since each fragment may be stored in more than one data module, in general there will be more than one way of reconstructing a complete and non-redundant copy of the logical database from the collection of stored fragments. For example, if fragment Fi is stored on data modules DMi and DMj as SFi,j and SFi,k, then there are at least two ways of reconstructing a complete, non-redundant copy of the logical database - one which includes SFi,j and another which includes SFi,k.

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 the RDDBS recognizes a specific set of materializations as "supported". A user logging in to the RDDBS is given access to the database through a specific supported materialization and, in general, repeated accesses to the RDDBS will result in assignment to the same materialization.

---

* The term materialization has been used previously by Chamberlin et al [CHAb] to refer to the process of constructing the contents of a virtual relation from stored relations.

The concept of supported materializations (called simply materializations in the remainder of this discussion) is central to the notion of consistency as defined in the next section. The logical consistency of the database is preserved within materializations. That is, the system will enforce a rigid form of consistency within each materialization and a looser form of consistency between materializations. As we shall see, this approach permits each user to see a logically consistent database without incurring excessive synchronization costs in performing updates

The make-up of each materialization is recorded in a table such as the one illustrated in Figure 3.2. * Another table indicates the materialization to which each user is assigned. The RDDBS will service a user's retrieval requests by accessing only the stored fragments listed for his assigned materialization. For updates more interaction between materializations is required. This will be described in Section 5.

The formulation of a materialization may need to be changed because of the failure of a supporting data module or because

* All system tables used to define the multi-module structure of the database are stored as ordinary data so that they may be stored with arbitrary dispersion and redundancy. Questions of centralization or decentralization of directories and dictionaries may therefore be delayed until database definition.

	F1	F2	F3	F4	F5
M1	1	1	1	2	3
M2	1	2	2	3	3
M3	2	3	3	2	3

TABLE ENTRY IS DM FROM WHICH MATERIALIZATION

M_I GETS FRAGMENT F_J.

Table of Fragment Assignments
Figure 3.2

load changes make another materialization definition more efficient. It is important that the redefinition process ensure the logical consistency of the new materialization. The specification of a mechanism for ensuring this consistency will be studied in future research.

4. Database Consistency

A central constraint on any update algorithm in a database system is the need to ensure that update activity does not violate the consistency of the database. We examined in a previous section (Section 2) the classical approach to this problem in centralized database systems. That approach entails locking those portions of a database that are affected by a transaction. We also examined in Section 2 several extensions of the classical approach that allowed it to be applied to RDDBSs. However, the locking solutions to the update problem in the redundant case were shown to be unsatisfactory in a number of ways.

The basic difficulty with locking schemes is that it is quite expensive to propagate locking information throughout an RDDBS. To circumvent this difficulty we suggested a redundant update solution based on a more efficient, non-locking protocol called the safe protocol.

As we have noted previously not all transactions may be run via the safe protocol. The constraint on which transactions can be handled via the safe protocol can be stated simply as follows:

> Let T1, T2,...,Tn be the class of transactions that are to be handled by the safe protocol. For any Ti in that class it must not be possible for Ti to interfere with any sub-class of the other T's in a manner that violates the consistency of the database.

Transaction that can be handled by the safe protocol are termed safe transactions.

In general there are many ways in which this partitioning of safe transactions vs. unsafe ones may be carried out. This topic is addressed in Section 5 below; several interesting and useful transaction classes are described there.

However no matter how the partitioning is done, one salient point remains:

> In order for the class of safe transactions to be useful to an RDDBS, it must be possible to define a predicate that tells whether or not a given transaction is safe. Without such a predicate there would be no way for a materialization to know when it could take advantage of the safe protocol, and all transactions would have to be handled via the locking protocol anyway.

> Furthermore, the predicate must be computable from data local to a single database module. The value of the safe protocol lies in avoiding multi-computer synchronization traffic; this value would be negated if the safety predicate -- a precursor to the use of the safe protocol -- were to require global communication in order to make its decision.

Consequently the study of local tests for the global condition of database consistency is central to our methodology.

Not surprisingly it turns out that the definition of the local predicates for transaction safety depends on the precise notion of the global property, database consistency, that is required in a given application. Before pursuing the classification of safe vs. unsafe transactions in detail it is advisable to have a clear understanding of the global property that must be preserved by the safe protocol.

The exploration of this global property is the topic of this section. The next section, Section 5, delves into the classification of safe vs. unsafe transactions, and the local tests that are needed to recognize the transaction classes.

4.1 Serializability -- the Global Property

We recall that all update algorithms that we have considered do consistency maintenance in two steps:

1. First each transaction is pre-checked, validated, and tested against specified database integrity constraints. The purpose of this step is to ensure that the transaction cannot violate database consistency by itself.

   This pre-checking of integrity conditions may always be done locally, i.e. within a single materialization. We shall not be further concerned in this paper with this step of the consistency maintenance procedure; hereafter we assume that all transactions have passed this step and are deemed acceptable individually.

2. The second step is to ensure that the transaction does not or cannot conflict with any other concurrent transaction in a manner that could violate database consistency. Essentially what this step does is

specify and enforce extra constraints above and beyond the integrity constraints that are checked in step (1). These extra constraints relate to the sequencing of transactions through the database that must hold if the database is to remain a faithful model of its domain.

A number of researchers have suggested that the correct sequencing constraint for many database applications is a property called "serializability" (e.g. [GRA], [HEW]).

Serializability dictates that the effect of a set of concurrent transactions on a database must be equivalent to some serial, non-overlapping sequence of those same transactions. For example, suppose T1, T2, T3, T4 are a set of transactions that have been run in a database, and that some or all of them were run concurrently. The effect of the transactions is "serializable" if and only if the state of the database following their completion is equivalent to a state that could have resulted from a completely one-at-a-time, non-overlapped sequence of the transactions, e.g.:

T1 then T3 then T2 then T4, or

T4 then T1 then T3 then T2, etc.

Serializability captures the intuitive notion that concurrent transactions not interfere with each other; if the result of some transactions is serializable then it appears that no transaction hurt the others.

4.2 Computing Serializability

The problem confronting us is that of constructing classes of transactions that may be run via the safe protocol. If we select serializability as the sequencing constraint that must be obeyed by all transactions, then this problem may be restated as follows:

> We must construct classes of transactions, $\underline{C}t$, with these properties:
>
> a. Any transactions in $\underline{C}t$ may be run concurrently without any danger of violating serializability; and
> b. It is possible to define local predicates that can tell whether a given transaction is a member of $\underline{C}t$.

Both of these properties involve the computation of serializability. Property (a) implies that serializability of the transactions in Ct was computed at some prior time, for instance at database design time. Property (b) requires that this pre-computation of serializability be accessible at run-time through local procedures.

We are concerned in this section with the first property; our task is to investigate methods for verifying whether or not a class of transactions obeys property (a), i.e. whether or not it is serializable.

Property (a), however, is not an easy property to verify. Given an arbitrary class of transactions T1, T2, T3, ..., Tn computation of their serializability seems to involve a combinatorial explosion. It appears that one must enumerate all the possible combinations in which the transactions may overlap, and for each combination compute whether the result is or is not serializable. In addition to the combinatorial difficulty, it has been observed by Bernstein and Papademitriou [BERN] that the computation of serializability for each arbitrary combination is itself an NP-complete problem (i.e., the computation is likely to grow exponentially with the number of transactions).

In order for our redundant update solution to be viable, therefore, we need to have mathematical tools that aid in computing the serializability of transactions. Also, we need a technique that allows increased reasoning power so as to help find new classes of serializable transactions.

4.3 The Graphic Technique for Determining Serializability

We have developed in conjunction with Bernstein [BERN] a technique that allows the serializability of a set of transactions to be determined easily. This technique is based on a graphic representation of relationships among members of a transaction class; the graphic representation is described in Figure 4.1. The central aspects of the graphic representation are as follows:

1. Each transaction, T, is decomposed into a read-set, RT, and a write-set, WT. The read-set, RT, is the set of all data items that T reads; i.e. it is the set of data items that serve as inputs to T, or that T uses in computing its results. The write-set, WT, is the set of all data items that T modifies.

T: R_T, W_T

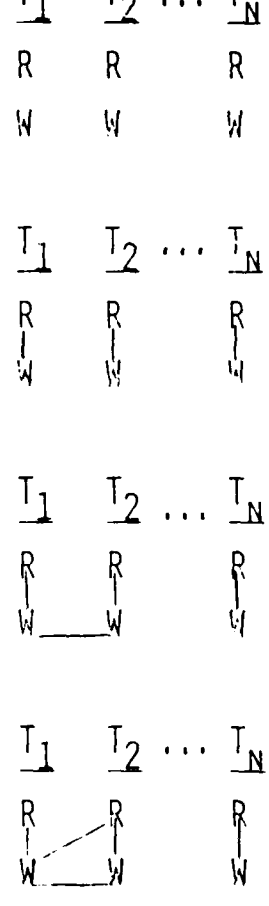
GRAPH OF A SET OF TRANSACTIONS -

1. CREATE NODES FOR THE READ SET AND WRITE SET OF EACH TRANSACTION.

2. DRAW AN ARC BETWEEN THE READ SET AND WRITE SET OF EACH TRANSACTION.

3. DRAW AN ARC BETWEEN TWO WRITE SETS IF THE SETS INTERSECT.

4. DRAW AN ARC BETWEEN A READ SET AND A WRITE SET IF THE SETS INTERSECT.



Graphic Representation of Transactions
Figure 4.1

For example, consider the PERSONNEL file in Figure 4.2, and the transaction T1:

T1: For the employee
whose EMPLOYEE-NUMBER = 123456789
Update POSITION to SECRETARY,
GRADE to III, and
SALARY to $10,000

T1's read-set, RT1, is the EMPLOYEE-NUMBER of one particular tuple. T1's write-set, WT1, is the POSITION, GRADE, and SALARY of that tuple.

2. Each transaction is represented in the graph by two nodes, one for its read-set and one for its write-set.

3. Arcs are drawn between nodes in the graph if and only if there is a relationship between the nodes such that the result in the database depends on the order in which the depicted reads or writes happen to occur. Operationally this means that:

Personnel

EMPLOYEE-NUMBER	NAME	POSITION	GRADE	SALARY ...
123456789	Smith	Receptionist	IV	$8,000

T1: For the employee whose EMPLOYEE-NUMBER = 123456789
Update POSITION to Secretary
GRADE to III
and SALARY to $10,000

T2: For all employees whose POSITION = Receptionist
and whose GRADE = IV
Update GRADE to V
and SALARY to $9,000

Personnel File and Two Transactions
Figure 4.2

a. Arcs are drawn between the read-set and write-set of each individual transaction, since presumably the values written depends on the values that were read. (This is step (2) in Figure 4.1). Such arcs are called vertical arcs.

b. Arcs are drawn between two write-sets if the sets intersect. For example, refering to the sample transaction T1 above, WT1 includes the SALARY field of a particular tuple; consider the transaction T2 in Figure 4.2 whose write-set, WT2, also includes that SALARY field. In general one can tell whether WT2 or WT1 happened last by the value left in the SALARY field. (This is step (3) in Figure 4.1.) Arcs drawn in this step are called horizontal arcs.

c. An arc is drawn between a read-set and a write-set if they intersect. For instance, T2's read-set, RT2, includes the POSITION field of the tuple affected by T1. The value of the POSITION field read by T2 depends on

whether RT2 was before WT1 or vice versa. Since the data written by any transaction depends in general on the values read, the results in the database in general will depend on the ordering of WT1 and RT2. (This is step (4) in Figure 4.1). Arcs drawn in this step are called diagonal arcs.

The graphic technique is useful because there are succinct relations between the topology of the graph and the serializability of the transactions represented therein. The main results we have developed to date with this technique are reviewed in the next section.

4.4 Results from the Graphic Technique

In this section we review the main results that we have developed using the graphic technique introduced in the previous section. These results relate the topology of a transaction graph to the serializability of the associated transactions. Using these results it is possible to determine the serializability of a set of transactions without incurring the combinatorial explosion that would be experienced without tools of this sort.

Result 1:

> In a non-redundant database, the presence of a cycle which includes a vertical arc (a V-cycle) in the transactions graph is a necessary condition for the transactions to be non-serializable. To state it conversely, a set of transactions is safe if its graph contains no V-cycles (see Figure 4.3).

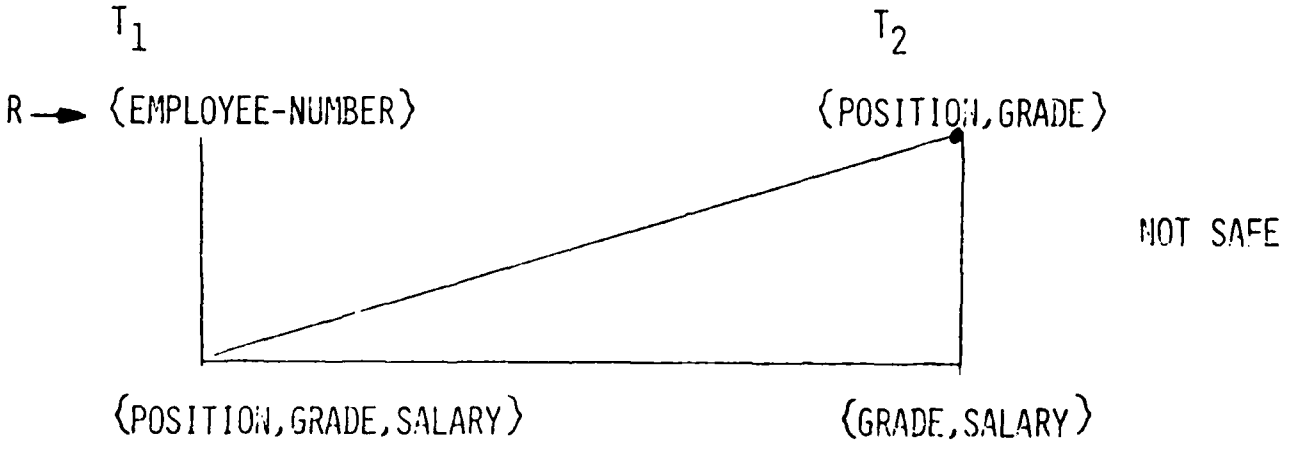
The presence of V-cycles in the graph is not a sufficient condition for non-serializability due to the possibility of dead transactions. A dead transaction is one whose result is

A SET OF TRANSACTION IS SAFE IF ITS GRAPH IS ACYCLIC.

T_1	T_2	T_3	
R	R	R	
W	W	W	SAFE

T_1	T_2	T_3	
R	R	R	
W	W	W	NOT SAFE

THE TRANSACTIONS FROM FIGURE 4.2



Examples of Transaction Graphs
Figure 4.3

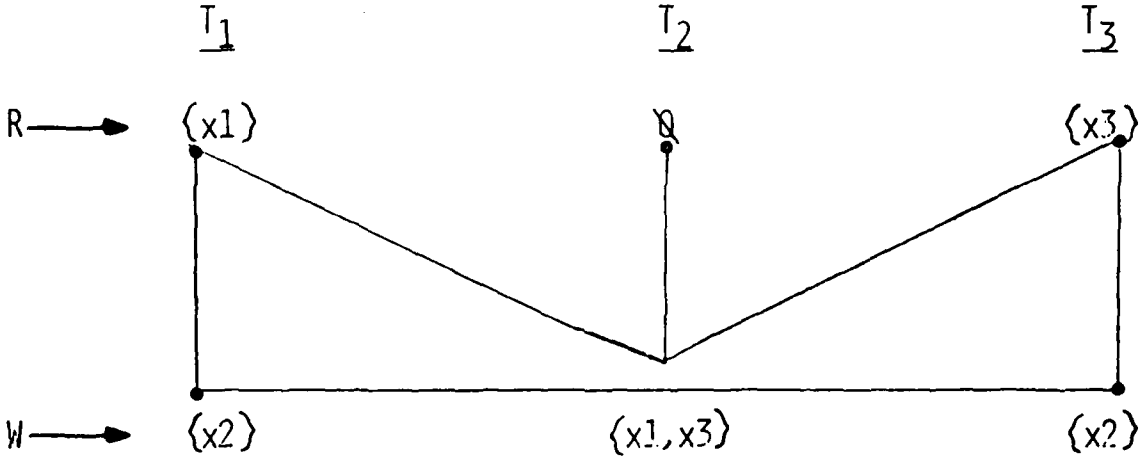
totally obliterated by some subsequent set of transactions. Figure 4.4 shows a V-cyclic graph that is serializable because its cycle contains a dead transaction. In that figure, either T1 or T3 will be dead depending on the order in which WT1 and WT3 occur. In any execution history, however, one or the other of them must be dead.

Result 2:

> In a non-redundant database without dead transactions, the presence of a V-cycle in the transaction graph is a necessary and sufficient condition for the transactions to be non-serializable.

Results (1) and (2) seem to indicate that the issue of dead transactions will play a large role in characterizations of safe classes of transactions. This is not so in many practical database applications, however, because of a phenomena called exhibited transactions.

Exhibited transactions are transactions that both update data in the database and exhibit information about the read-set to external observers. Exhibited dead transactions may be dead vis a vis their effects on the database state, but they are very much alive in terms of their effect on the external world.



V-cyclic Graph with Dead Transactions
Figure 4.4

As an example of exhibited dead transactions consider the database and transactions in Figure 4.5. The database includes a file that contains the assignments of Navy personnel to ships. For each tuple in the file, that person is available for reassignment if the AVAILABLE attribute equals "Yes". The two transactions in the example may have been initiated by two personnel officers and are simply trying to find an available seaman and assign him to a ship. T1 is trying to assign the seaman to the ENTERPRISE while T2 wishes to assign him to the JFK.

There is of course a V-cycle in the graph of T1 and T2 (see Figure 4.6). However, in any execution history one of the two transaction is assured of being dead; thus running T1 and T2 concurrently doesn't violate serializability, and it would appear that T1 and T2 could be run via the safe protocol.

But if T1 and T2 are run concurrently, anomalous database behavior may result: both personnel officers will think that he had assigned a seaman to a ship; however only one of them may have actually succeeded in doing so. The personnel officer whose transaction became dead will have failed to make the assignment.

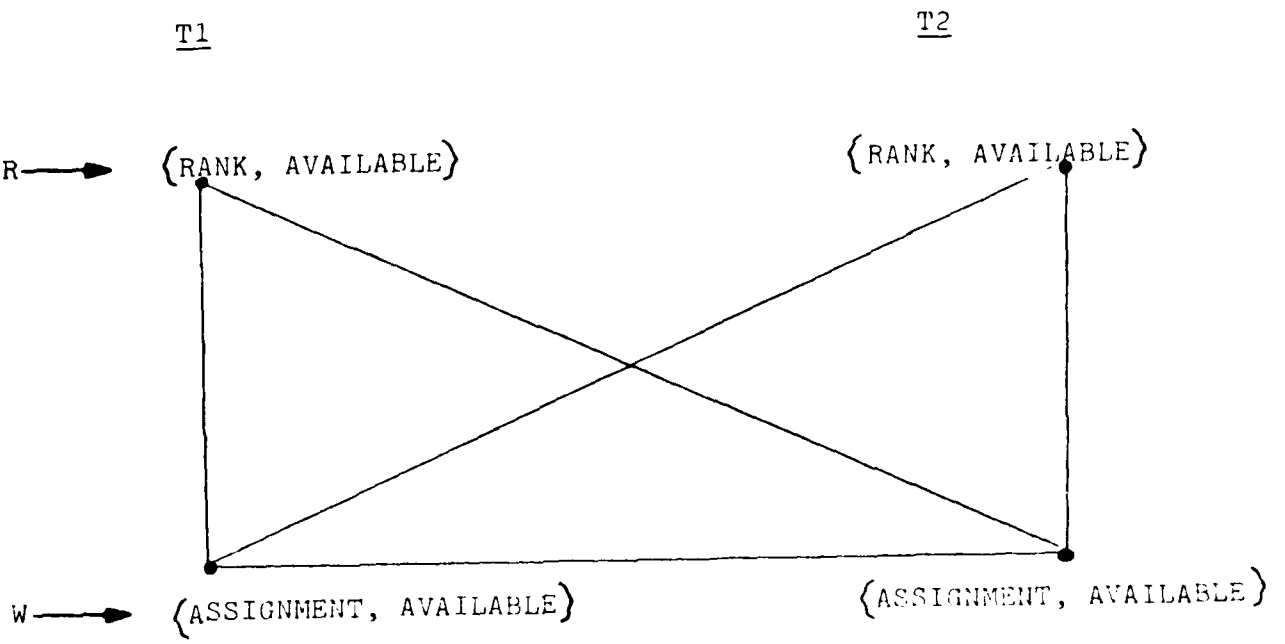
PERSONNEL

Soc-Sec-Number	Name	Rank	Available	Assignment
123456789	Jones	Seaman	Yes	Norfolk, VA.
234567890	Brown	Seaman	No	WASP
345678901	Gray	Seaman	Yes	San Diego, CA.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

T1: For any person with RANK = SEAMAN,
and AVAILABLE = Yes
Update ASSIGNMENT to ENTERPRISE,
and AVAILABLE to NO.

T2: For any person with RANK = SEAMAN,
and AVAILABLE = YES
Update ASSIGNMENT to JFK,
and AVAILABLE to NO.

Exhibited Dead Transactions
Figure 4.5



Graph of Transactions in Figure 4.5
Figure 4.6

To avoid this anomaly we postulate that data exhibited to external observers is conceptually part of the write-set of the transaction. We assume that in general information presented to external observers cannot be obliterated by later transactions. Hence, by including the external observers in the conceptual database, exhibited transactions can never be dead.

This brings us to Result 3:

> In a non-redundant database where all transactions are exhibited, the presence of a V-cycle in the transaction graph is a necessary and sufficient condition for non-serializability.
>
> Result 3 follows directly from Result 2.

It is interesting to consider at this point the role of retrievals in this scheme. We model retrievals in the graphic representation by transactions called observer transactions. An observer transaction is one that reads data that may be read or written by normal transactions, but writes data that may never be touched by any other transactions. The write-set of observer transactions represents possibly the terminal of the user who performed the retrieval or some other external record of the retrieval.

Arbitrary observer transactions are ones whose read-set includes arbitrary data items. In particular, the read-set may intersect the write-sets of all the other transactions. Arbitrary observer transactions model the most general retrievals possible.

Result 4:

> In a non-redundant database that permits arbitrary observer transactions, a diagonal arc is a necessary condition for non-serializability. That is, a set of transactions is safe in the presence of arbitrary observer transactions if its graph contains no diagonal arcs.

The results that we have presented so far apply to non-redundant databases only. When applied to redundant databases the results break down due to the possibility of transactions arriving at a materialization out of order.

Suppose T1 and T2 are transactions and that T1 is before T2 in the correct global sequencing of transactions. If T2 arrives at a materialization before T1 does, then to an observer at materialization, the database may appear to go "backwards".

An example of this sort is presented in Figure 4.7. The figure shows two transactions, T1 and T2, that are initiated at arbitrary materializations, and a set of observer transactions, Qi, that are initiated at the materialization, MO. T1 and T2 each simply sets a corresponding variable, Vi, to YES, and sets the variable LAST to its own name; LAST thus reflects the last transactions to set a Vi to YES. The Qi are observer transactions that print out two types of information: (1) they print out the value of LAST, and (2) they print out the names of all Vi that currently equal YES.

A graph of these transactions appears in Figure 4.7. You may note that this situation conforms to the hypothesis of Result (4), except that the database here is assumed to be redundant.

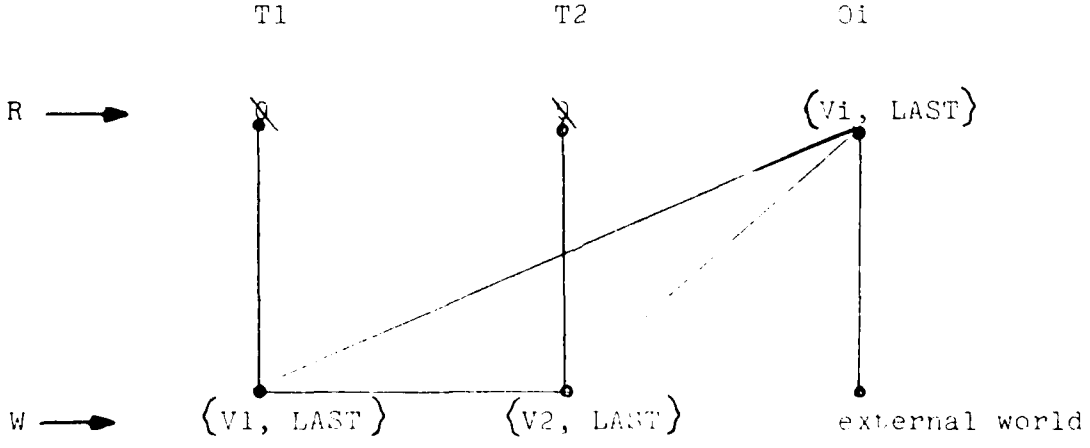
A plausible integrity constraint on the operation of this database is:

> For two transactions Qa and Qb where Qa is before Qb, if more Vi are YES in the results of Qb then in the result of Qa, then the value of LAST must be different in the two transactions.

T1: Set V1: = YES, LAST: = T1.

T2: Set V2: = YES, LAST: = T2.

Oi: Print LAST, and all Vi that are equal YES.



Anomalous Case in Redundant Environment
Figure 4.7

This constraint simply states that no transaction may set a Vi to YES without also changing LAST.

In a redundant database, if T1 and T2 are run via the safe protocol, this integrity constraint may not be maintained. Suppose that in the correct global sequencing of T1 and T2, T1 is before T2, but that T2 is received by M0 before T1. It is possible then that transactions will be run at M0 in the following order:

T2 then Qa then T1 then Qb

In this case, Qa would observe the effects of T2 and would print:

LAST=T2, LIST OF YESES=V2

Qb would observe the effects of both T1 and T2. But since T1 is before T2 in the global sequencing, the effect of T1 on the database must not undo any of T2's effects. (This is assured by the time stamping employed by the safe protocol).

Thus, although V1 is set to YES as a result of T1's arrival at M0, LAST is not modified at that time. Consequently what Qb prints out is:

LAST=T2, LIST OF YESES=V1, V2

The LIST OF YESES is longer in Ob than in Oa but LAST is not changed: the integrity constraint is not upheld.

Result 5:

In a redundant database, the presence of any cycle (V-cycle or otherwise) is necessary for non-serializability.

In the example of Figure 4.7, had the write-sets of T1 and T2 not intersected via the variable, LAST, the anomalous behavior would not have been possible.

4.5 Conclusion

Solutions to the redundant update problem must ensure that the consistency of the database cannot be violated by update activity. To do so the update algorithm must pre-check each transaction to test whether the transaction by itself violates any integrity constraints of the database. Then the algorithm must apply sequencing constraints to the transaction to ensure that multiple transactions cannot interfere with each other in a manner that violates database consistency.

For many database applications the necessary sequencing constraints is serializability. Consequently in our approach to the redundant update problem, we need to find classes of transactions that may be run concurrently without violating serializability.

Verifying the serializability of arbitrary classes of transactions is a difficult problem, though, that appears to entail a combinatorial explosion of computation. To circumvent this combinatorial problem we introduced a graphic representation of the relations among concurrent transactions. In this graphic representation the question of serializability is

reduced to simpler questions concerning the topology of the graph. A number of preliminary results were presented that relate certain types of graphs to the serializability of various classes of transactions.

A major thrust of future research will be to explore the use of this graphic technique further. Among the issues to be investigated are these:

1. In terms of transaction graphs, what condition is necessary and sufficient for serializability of a class of transaction in a redundant database? What is the condition if arbitrary observer transactions are allowed? What is it with various classes of restricted observer transactions?

2. How can the graphic technique be modified to permit better set resolution? As described here, an arc means merely that there is some intersection between the sets; at this level of resolution we cannot reason about practically overlapping portions of read or write sets.

   We have already seen one instance where it was necessary to distinguish between total and partial intersections of write-sets. (This was with regard to

dead transactions). We expect that the need for finer resolution will recur in other instances.

There are other issues we expect to study in addition to the graphic techniques per se. Among these are:

3. Throughout our preliminary work we have assumed that all transactions perform arbitrary computations based on their read-sets in order to arrive at values for their write-sets. That is, we have not attempted to look inside transactions to see what they do. What can be done to weaken this assumption? What is the effect of considering limited types of transactions? Can this be modelled in our graphic representation and would that be helpful?

4. Observer transactions add many arcs in the transactions graph representation and thus seem to reduce the amount of concurrency possible in a redundant database. What can be done to ameliorate this difficulty? One idea is to let observer transactions advise the system as to whether they really require totally consistent data. Possibly in certain applications there are natural measures of the extent to which the database is not consistent and some obser-

ver transaction can tolerate up to a certain amount of inconsistency.

In the next Section we take up the other side of the two part safety problem. That is, given a statement of the condition which assures that a network-wide set of transactions is safe, how can a transaction be tested at a single data module to determine if it is a member of that set?

5. Local Test for Safety

Having explored the characteristics of the network-wide set of transactions which can lead to database inconsistencies, we now move to consider efficient means for controlling the transaction mutual interference which causes inconsistency. Specifically we wish to detect a transaction which may violate the safety of the set of concurrent transactions without actually knowing what other transactions are executing. We seek a safety test, Sm(T), which can be applied to a transaction T initiated in materialization M and which can determine if T may violate network-wide safety by considering only that data and those other transactions known currently in M. If Sm(T) is true then the results of running T in M can be transmitted to the other materializations in the system via the safe (unsynchronized) protocol. Otherwise, a synchronizing protocol must be employed.

The construction of local safety tests (Sm's) is a data base design activity in which the tests to be applied at all materializations are established in a coordinated fashion. This design step permits each materialization to know what trans-

actions could be running concurrently in other materializations. Because of this limited global knowledge about the set of transactions running throughout the network a materialization may determine that a given transaction could not violate the safety of the set of concurrent transactions.

In section 5.1 the general form of Sm is described. Then, section 5.2 describes the process of coordinating the tests at all materializations so that the set of transaction they permit to run concurrently is safe. Section 5.3 sketches some example safety tests and section 5.4 suggests topics in the area of local safety tests which could be explored in the contemplated research effort.

5.1 Form of the Local Safety Test

The safety test considered here involve the definition of classes of transactions as a database design step. Specific classes are established for each materialization. The safety test at materialization M simply asks if the transaction in question is a member of one of M's class. If so, the transaction passes; otherwise it does not.

A class C is defined by a read-set Rc and a write-set Wc. The class is the set of transaction which reads only from Rc and writes only to Wc. That is

$$C = T \quad Rt \quad Rc \text{ and } Wt \quad Wc$$

Classes are defined for each materialization and are denoted by Ci,j to designate the jth class of the ith materialization.

The safety test for materialization M is defined as follows:

That is, the safety test is passed for a transaction T if T is a member of one of M's transaction classes. If the test is passed then the results of running T in M are passed to the other materializations via the safe protocol.

The set of safety tests defined in a database design is called the safe configuration for that design. The next section considers what constraints a safe configuration must satisfy in order to insure that inconsistencies do not arise.

5.2 Constraints on Safe Configurations

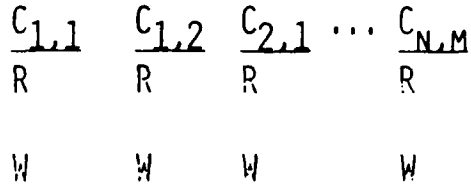
The intent of the safety test scheme just described is to localize transaction conflict within materializations where it is inexpensive to detect via locking and to avoid conflict among transactions in different materializations where it is expensive to detect. Hence, the constraint we define on safe configurations will permit conflict within a materialization but will prohibit conflict in sets of transactions executing in different materializations.

The mechanism for detecting unacceptable conflicts in a safe configuration is comparable to the graphic technique introduced in section 4 for discovering non-serializable sets of

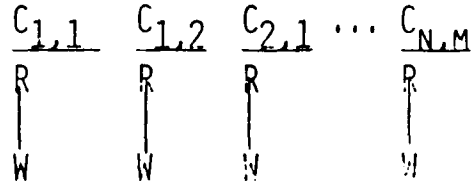
Detecting Conflict in a Safe Configuration

1. Form Basic Graph of Safe Configuration.

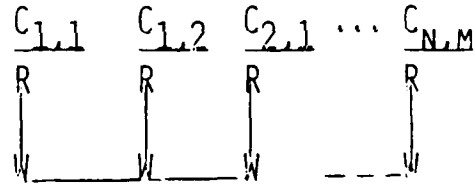
   A. Create Nodes for Read Set and Write Set of Each Class.



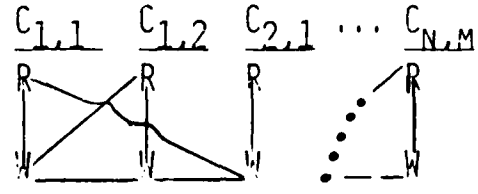
   B. Draw an Arc between the Read Set and Write Set of Each Class.



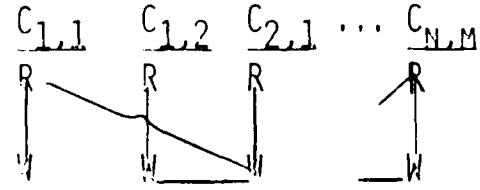
   C. Draw an Arc between two Write Sets if the Sets Intersect.



   D. Draw an Arc between a Read Set and a Write Set if They Intersect.



2. Eliminate Arcs created in steps C and D for Classes Defined for Same Materialization.



3. Check Graph for Consistency Condition

Graphic Technique for Classes of Transactions
Figure 5.1

transactions. This method is shown in figure 5.1. A graph is formed for classes using the algorithm employed for transactions except that no arcs are drawn between classes defined for the same materialization. Locks employed within the materialization will prevent transactions with intersections on either write set from running concurrently. Hence, these paths of potential conflict cannot be followed.

The graph drawn according to the rules of figure 5.1 represents a generator for sets of transactions which can exhibit the same sorts of connections as their parent classes. If we wish to impose topological restrictions on the transaction graphs then we hypothesize that it is necessary and sufficient to impose those same restrictions on the graph of their classes. For example, to avoid cyclic transaction graphs it is only necessary to avoid cycles in the graph produced by figure 5.1. A similar result applies to avoiding diagonal and horizontal arcs. Hence, in order to constrain a safe configuration to those class definitions which avoid inconsistencies, one applies the desired transaction consistency rule to the class graph. The safe configuration is acceptable if and only if that consistency rule is not violated

5.3 Examples of Safe Configurations

This section attempts to suggest the power and flexibility of the local safety test approach by briefly describing examples of its use.

Primary site approach - The primary site method of Alsberg et al [ALS] which was mentioned in section 1 is a special case of a local safety test in which all transactions pass the safety test at the primary materializations and all fail the test elsewhere. Formally:

$\underline{C}p,1$: Read-set = complete database
Write-set = complete database
where P is primary materialization

$\underline{C}m,1$: Read-set = 0
Write-set = 0
where M P

The graph of this configuration has no horizontal or diagonal arcs since all classes are defined for only one materialization.

Simple updates - In many practical database applications the bulk of the updates involve the identification of a record by

a certain field or fields (e.g. social security number) and the modification of other fields (e.g. salary, department, job title). One means of handling this situation as a safe configuration is to have the database designer partition the complete database into two parts, say Rs and Ws. Then the same class is defined for all materializations. It specifies transactions which read only from Rs and write only to Ws. Formally

$$C_{m,1}: \quad \text{Read-set} = R_s$$
$$\text{Write-set} = W_s$$
$$\text{for all } M$$

The graph of this configuration contains horizontal arcs but no diagonals.

Local interests - A phenomenon which is expected to be common in the use of geographically dispersed distributed database systems is the predominant interest of users in the local portion of a global database. That is, while there is some reason to integrate the data across widely separated groups most traffic clusters around the local interests of the respective groups. It is clearly desirable to avoid global synchronization of these locally oriented transactions.

[illegible] for example, a distributed version of a naval [illegible] in which stored fragments represent [illegible]

AD-A125 164 AN APPROACH TO UPDATING IN A REDUNDANT DISTRIBUTED DATA BASE ENVIRONMENT(U) COMPUTER CORP OF AMERICA CAMBRIDGE MA J B ROTHNIE ET AL. 15 FEB 77 CCA-77-01 2/2

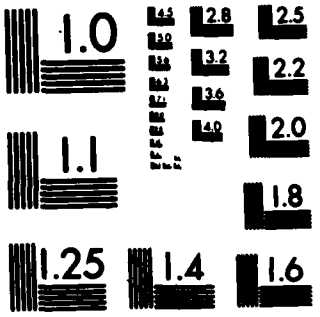
UNCLASSIFIED N00039-77-C-0074 F/G 9/2 NL

END

DATE FILMED

83

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

personnel on each ship are stored on that ship's local data module (as well as elsewhere). The problem is to define a safe configuration which permits direct manipulation of these local records without external synchronization.

To accomplish this, each ship (for example, the JFK) will have the following class defined:

CJFK,1 : Read-set = all personnel data where assigned ship is JFK

Write-set = same

5.4 Examples of Topics to be Explored

The concept of local testing of the global consistency condition is rich in interesting sub-topics and in practical significance. We suggest a careful analysis of questions in this area including the following:

Proof of safe configuration constraint rule - We have hypothesized in this section that a graph constructed as described in figure 5.1 can be tested for the global consistency constraint exactly as a transaction graph. This idea is of such central importance that it requires a formal proof.

Database design - Database design is already a difficult problem with the variety of indexing and accessing alternatives offered by current systems. Distribution and redundancy dramatically increase the complexity of that job and the introduction of safe configurations seems likely to place this task beyond the scope of intuitive methods. We will explore means of optimizing (either formally or heuristically) those portions of the design problem which relate to the redundant distributed environment. This effort will be aimed at the construction of a practical tool to aid a designer in this process.

Impact of global conditions - The safe configurations which can be employed for a given data base depend upon the type of consistency which the system is seeking to achieve. We intend to explore the impact of the choice of this condition on the construction of local tests and on database design.

Materialization groups - The presentation in this section dealt with two levels of synchronization - within a single materialization and within the complete network of materializations. It is possible to define a multi-level synchronization structure in which transactions which cannot be declared safe within the scope of a single materialization can be declared safe by synchronizing with several but not all materializations. We will explore the circumstances under which this is possible and consider the performance impact of this additional dimension of flexibility.

Overlapping materializations - When two materializations share a stored fragment there are a number of updating impacts to be considered. The underlying issue is that updating that portion of one materialization also updates the other one. This leads to potentially efficiency gains and to potential inconsistency problems. We will investigate these impacts in detail and consider the effects of performance and of database design.

References

[ALS]
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.)

[BER]
Berry, J. E.; and Cook, C. M. "Managing Knowledge as a Corporate Resource", unpublished paper, May 28, 1976.

[BERN]
Bernstein, P. A.; and Papadimitriou, C. Private communication, November 1976.

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

[CCA]
Computer Corporation of America "A Distributed Database Management System for Command and Control Applications", proposal submitted to the Advanced Research Projects Agency, July 12, 1976.

[CHAa]
Chamberlin, D. D.; Boyce, R. F.; Traiger, I. L. "A Deadlock-free Scheme for Resource Locking in a Database Environment", Information Processing 74, Proceedings AFIPS Conference, North Holland Publishing Company, Amsterdam, The Netherlands, 1974.

[CHAb]
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.

[CHU]

Chu, Wesley W. "Optimal File Allocation in a an Information Network", Proceedings 1972 SJCC, AFIPS Press, 1972.

[COD]

Codd, E. F. "Further Normalization of the Database Relational Model", in Database Systems (Courant Computer Science Symposium 6, R. Rustin ed.), Prentice-Hall, 1972, pp. 3364.

[DIJ]

Dijkstra, E.W. "The Structure of 'THE' Multiprogramming System", CACM, Vol. 11, No. 5, May 1968, pp. 341346.

[ESW]

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]

Fry, J. P.; and Sibley, E. H. "Evolution of Database Management Systems", Computing Surveys, Vol. 8, No. 1, March 1976, pp. 742.

[GRA]

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

[HEW]

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

[HOA]

Hoare, C. A. R. "Monitors. An Operating System Structuring Concept", CACM Vol. 17, No. 10, October 1974, pp 549557.

[LEV]

Levin, K. D.; and Morgan, H. L. "Dynamic File Assignment in Computer Networks Under Varying Access Request Patterns", Technical Report No. 750401, Department of Decision Sciences, The Wharton School, University of Pennsylvania, April 1975.

[MAH]

Mahmoud, S.; and Riordan, J. S. "Optional Allocation of Resources in Distributed Information Networks", TODS, Vol. 1, No. 1, March 1976, pp. 6678.

[ROS]

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

[ROT]

Rothnie, J. B. "Evaluating Inter-Entry Retrieval Expressions in a Database Management System", Proceedings AFIPS National Computer Conference, AFIPS Press, Vol. 44, 1975.

[SCH]

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.

[SIB]

Sibley, E. H. "The Development of Database Technology" Computer Surveys, Vol. 8, No. 1, March 1976, pp. 15.

[STO]

Stonebraker, M.; and Neuhold, E. "A Distributed Database Version of INGRES", unpublished paper, November 6, 1976.

[THOa]

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

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

[WHI]
Whitney, V. K. W. "A Study of Optimal File Assignment and Communication Network Configuration in Remote-Access Computer Message Processing and Communication Systems" Ph.D. Dissertation, Department of Electrical Engineering and College of Engineering, University of Michigan, September, 1970.