12-1986

Horizontal Fragmentation and Allocation in a Distributed Database for Cost Minimization with Reliability and Space Constraints

Elmo Loren Ivey
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/1334

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
HORIZONTAL FRAGMENTATION AND ALLOCATION
IN A DISTRIBUTED DATABASE FOR COST
MINIMIZATION WITH RELIABILITY
AND SPACE CONSTRAINTS

by

Elmo Loren Ivey

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
December 1986
In this paper a method for horizontal fragmentation of a relational database is presented. A method is given for calculating the benefit or cost savings obtained by allocating a given fragment to a site, independently for each site.

To obtain optimal benefit each fragment is allocated to all sites where there is a positive benefit. This may result in the allocation of some fragments to multiple sites.

Finally, a method is given for removing or reallocating certain fragments from sites with insufficient storage. Also, a method is given for allocating fragments to multiple sites to meet constraints on the required reliability of applications of the database.
ACKNOWLEDGMENTS

I would like to express my deep appreciation to Dr. Dalia Motzkin for her help and guidance in the preparation of this paper.

Elmo Loren Ivey
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University
Microfilms
International
300 N. Zeib Road
Ann Arbor, MI 48106

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Ivey, Elmo Loren

HORIZONTAL FRAGMENTATION AND ALLOCATION IN A DISTRIBUTED DATABASE FOR COST MINIMIZATION WITH RELIABILITY AND SPACE CONSTRAINTS

Western Michigan University

M.S. 1986

University Microfilms International

300 N. Zeeb Road, Ann Arbor, MI 48106
# TABLE OF CONTENTS

ACKNOWLEDGMENTS ........................................... ii
LIST OF TABLES ........................................... vi
LIST OF FIGURES ........................................... vii

CHAPTER

I. INTRODUCTION ........................................... 1
   Statement of the Problem ................................. 1
   Purpose of the Paper .................................... 2
   Assumptions and Limitations. ............................ 3
   Organization of the Paper ................................. 3

II. REVIEW OF RELATED LITERATURE .......................... 5

III. OVERVIEW OF THE METHOD ............................... 9

IV. DEFINING SEGMENTS ...................................... 12
   Explanation and Examples ................................ 12
   Algorithm .............................................. 15

V. DEFINING FRAGMENTS ..................................... 22
   Explanation and Examples ................................ 22
   Algorithm .............................................. 24

VI. ALLOCATION OF FRAGMENTS ............................. 26
   Costs .................................................... 26
   Assumptions and Simplifications ......................... 26
   Explanation ............................................. 27
   Algorithm .............................................. 28
   Example ................................................ 29
# Table of Contents—Continued

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraints</td>
<td>32</td>
</tr>
<tr>
<td>Space</td>
<td>33</td>
</tr>
<tr>
<td>Reliability</td>
<td>35</td>
</tr>
<tr>
<td>VII. IMPLEMENTATION</td>
<td>40</td>
</tr>
<tr>
<td>Input and Output</td>
<td>40</td>
</tr>
<tr>
<td>Define Segments</td>
<td>42</td>
</tr>
<tr>
<td>Relational Operators</td>
<td>42</td>
</tr>
<tr>
<td>Define Fragments</td>
<td>44</td>
</tr>
<tr>
<td>Method</td>
<td>44</td>
</tr>
<tr>
<td>Getting Data for Allocation</td>
<td>45</td>
</tr>
<tr>
<td>Algorithms</td>
<td>47</td>
</tr>
<tr>
<td>Allocation</td>
<td>47</td>
</tr>
<tr>
<td>Benefits</td>
<td>47</td>
</tr>
<tr>
<td>Space Constraints</td>
<td>48</td>
</tr>
<tr>
<td>Reliability Constraints</td>
<td>49</td>
</tr>
<tr>
<td>VIII. COMPLEXITY</td>
<td>50</td>
</tr>
<tr>
<td>Define Segments</td>
<td>50</td>
</tr>
<tr>
<td>Define Fragments</td>
<td>50</td>
</tr>
<tr>
<td>Get Costs</td>
<td>51</td>
</tr>
<tr>
<td>Meet Space Constraints</td>
<td>51</td>
</tr>
<tr>
<td>Meet Reliability Constraints</td>
<td>51</td>
</tr>
<tr>
<td>IX. CONCLUSION</td>
<td>52</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>54</td>
</tr>
</tbody>
</table>
Table of Contents--Continued

A. Sample Runs .................................. 54
B. Detailed Algorithm ............................. 61
BIBLIOGRAPHY ..................................... 83
LIST OF TABLES

1. A Sample Database ........................................ 9
2. The Applications of the Database ....................... 12
3. The Applications Expressed as the Relational Operators SELECT and SEMIJOIN ............ 14
4. The Segments of the Global Database .................... 16
5. The Fragments of the Global Database .................... 22
6. Statistical Data for Determining the Benefit of Each Fragment to Each Site ................ 29
7. The Benefit of Allocating the Fragments to the Sites .................................................... 31
# LIST OF FIGURES

1. An Algorithm for Getting the Segments of Each relation Determined by the Applications of the Global Database 15

2. An Algorithm for Splitting the Segments in Pairwise Disjoint Fragments 25

3. An Algorithm for Allocating the Fragments to All Sites Where There is a Positive Benefit 28

4. The Allocation of Fragments to the Sites 32

5. An Algorithm for Removing Fragments From Sites With Insufficient Space 34

6. The Allocation of Fragments Where Some Fragments With a Calculated Positive Benefit are Not Allocated Due to Insufficient Space 35

7. An Algorithm for Determining the Probability of Failure for Each Fragment 36

8. An Algorithm to Allocate Multiple Copies of Fragments to Meet Constraints on the Required Reliability of the Applications 37

9. The Allocation of the Fragments Where Some Fragments With a Negative Calculated Benefit Are Allocated to Meet the Reliability Constraints 38

10. An Algorithm to Output a Message If Any Fragment Is Not Allocated at All 39

11. An algorithm to Assign Statistical Data to the Fragments 46

12. An Algorithm to Check That All Records in a Relation Belong to Some Fragment 47

13. An Algorithm to Find the Number of Remote Updates Issued Per Day to Fragment #1 at Site #J 48

14. An Algorithm to Get the Space Used at Site #1 49
Chapter I

Introduction

Statement of the Problem

A distributed database has several advantages over a centralized database for organizations which are spread over several sites. Files stored at the site of use can be accessed faster than files stored at a distant site. A distributed database also provides greater reliability and availability if multiple copies of a fragment of the database are stored. And a distributed database is able to respond more quickly to changes in technology or user requirements than a centralized database.

With the decrease in the cost of smaller computers it may now be less expensive to store data on several smaller computers than on one large computer.

With a distributed database there is the problem of how to partition the database and distribute it over the sites to gain maximum advantage while keeping costs to a minimum.

Storing multiple copies increases availability and reliability and may reduce communication costs for data retrieval since a site will have much of the data it uses stored locally, but communication costs are increased for
updates since every site with a copy of the data must have its copy modified every time an update is performed.

In this paper a method is presented for breaking the global database into fragments and calculating the benefit, that is, the cost savings, that can be obtained by allocating a given fragment to a given site. Then by allocating all fragments to all sites where the benefit is positive a globally optimal cost reduction is achieved. A method is then presented for reducing the number of allocations, or reallocating some fragments, to meet constraints on the available storage space at each site. A method is also presented for increasing the number of allocations to meet constraints on the required reliability of each application.

Purpose of the Paper

This paper is based on the ideas and algorithms presented by Dalia Motzkin (Motzkin, 1986). This paper presents her design methodology for a distributed database management system and presents the implementation of a design tool to accomplish this design. The major objective is the implementation of Motzkin's design methodology.

The object of the design is to take advantage of the reduction in communication costs possible by distributing the global database over the sites using it and to take
advantage of the increased availability and reliability possible when multiple copies of some files are allocated. This paper gives a method for minimizing the costs while meeting constraints on the storage space available at each site and the required reliability of each application. It is assumed that storage is limited as would be the case with a network of minicomputers or microcomputers.

**Assumptions and Limitations**

Some assumptions and simplifications are made to reduce the number of input parameters that must be estimated. An algorithm is practical only if the input is of a manageable size. Heuristic algorithms are used to meet the space and reliability constraints to obtain realistic time complexity.

**Organization of the Paper**

1. A brief description of some of the related literature is given.
2. An overview of the method for the logical design of a distributed database is given.
3. The method of breaking the global database up into segments based on the applications to the global database is given.
4. Then it is shown how to break these segments into pairwise disjoint fragments for distribution over the network of sites.

5. Next the allocation of the fragments to the sites based on the cost of allocation is presented, followed by modifications to the allocation scheme based on storage space and reliability constraints.

6. The implementation of this method is discussed, followed by a discussion of the complexity of the algorithms used.

7. Then the conclusion is followed by appendices with sample runs and a detailed algorithm of the program used.
Ceri and several of his colleagues have published several papers on the file allocation problem, FAP. They refer to several previous investigations of the FAP and conclude that all of them suffer from computational complexity and/or number of parameters that need to be estimated for the input. This makes them impractical in real life (Ceri, Martella, Pelagatti, Deen & Hammersley, 1980).

They work out models for both redundant and non-redundant file allocation with storage space as a constraint, but not a cost. They then show the equivalence in both cases to the knapsack problem. They consider only predefined transactions, a unit local access cost and unit remote access cost. In the nonredundant case retrievals and updates are not distinguished (Ceri, Martella & Pelagatti, 1982).

They discuss the use of predicates and minterms in the definition of the segments used by the applications (Ceri, Negri & Pelagatti, 1982).

They also develop a form of linear integer zero-one programming problem for optimizing the FAP with non-redundant allocation. Redundant file allocation is
added as a heuristic post optimization (Ceri, Navathe & Wiederhold, 1984).

Chang and colleagues use a structured decomposition method for decomposing a database relation. Both horizontal and vertical decomposition are accomplished using the decomposition operators: vertical concatenation, stacking, horizontal concatenation, qualified vertical concatenation, and qualified stacking.

These operators are used iteratively to get a decomposition structure, which is a directed graph where the nodes are the composite or elementary relations. An elementary relation is defined as a relation in third normal form and a composite relation as a relation that can be decomposed into elementary relations. Arcs are directed from a relation to the relations decomposed from it (Chang & Cheng, 1980; Chang & Liu, 1981, 1982).

Dowdy and Foster (1982) compare several models for the FAP for features and tractability. They conclude that any detailed analysis involving all the components at one time will be intractable. And they point out that intuitive heuristics sometimes give poor results.

Irani and Khabbaz study the combined FAP and network design problem for network design topologies other than the tree or fully connected (Irani & Khabbaz, 1979, 1981).

Lin and Liu (1981) investigate the design of a
distributed double-loop data network for very large on-line databases. They emphasize the hardware used.

Navathe and colleagues investigate vertical partitioning in a database rather than horizontal partitioning. In this case the "partitions" are not necessarily disjoint (Navathe, Ceri, Wiederhold & Dou, 1984).

Rakes and colleagues give a model for FAP as an integer zero-one programming model where file redundancy is enforced to insure adequate reliability and they give a heuristic, for reducing the complexity, that assumes all transmission costs to be equal (Rakes, Franz & Se, 1984).

Reddy (1981) discusses the advantage of distributed databases over centralized databases and he discusses the differing network topologies and directory systems. He gives a file allocation policy for minimizing average access time.

Many of the ideas in this paper were taken from Ceri and his colleagues, e.g., global optimization viewpoint, storage limitation constraints, using redundant allocation to improve reliability, and restricting fragmentation to horizontal fragmentation.

The idea of creating pairwise disjoint fragments from possibly overlapping segments was presented in Rakes, Franz & Sen (1984).

However, this paper takes a new and different
approach to figuring the costs, where benefit (least cost) is calculated independantly for each site. Also, both storage space and reliability constraints are taken into account.
CHAPTER III

OVERVIEW OF THE METHOD

We will assume a fully defined relational database, see Table 1, with a fully connected network topology.

Table 1.
A Sample Database.

<table>
<thead>
<tr>
<th>STORE</th>
<th>STORE#</th>
<th>ADDRESS</th>
<th>EQUIPMENT</th>
<th>INVENTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>APPLE</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>ATARI</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>COMMODORE</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>APPLE</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>COMPAQ</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>IBM</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PLAINFIELD</td>
<td>KAYPRO</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PLAINFIELD</td>
<td>ZENITH</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SALESMAN</th>
<th>STORE#</th>
<th>EXPERTISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>1</td>
<td>APPLE</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>ATARI</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>COMMODORE</td>
</tr>
</tbody>
</table>
The design process is divided into two phases.

1. Fragment definition

2. Fragment allocation
During the first phase the global database is split into fragments which meet the following conditions:

1. The fragments are pairwise disjoint.
2. The union of all fragments of a given relation is equal to that relation.
3. Given an application, the fragment is either completely required by the application or not required at all.

Cost, storage and reliability are not taken into consideration during the first phase. This phase merely insures the fragments cover the entire global database and that the allocation can be accomplished without allocating records to sites where they will be useless.

In this phase, first the database is segmented according to the data needed for each application at each site, then the segments are further broken down to produce a set of pairwise disjoint fragments covering the entire database.

During the second phase statistics on the type of use (retrieval or update) and their frequencies along with constant cost factors are used to allocate the fragments to the sites constrained by the amount of storage space available at each site and the required reliability for each application.
CHAPTER IV

DEFINING SEGMENTS

Explanation and Example

To distribute the database efficiently among the sites, each site must determine all the applications of the database that will be used at that site. See Table 2.

Table 2
The Applications of the Database.

<table>
<thead>
<tr>
<th>SITE</th>
<th>APPLICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>A1</td>
</tr>
<tr>
<td>S1</td>
<td>A2</td>
</tr>
<tr>
<td>S1</td>
<td>A3</td>
</tr>
<tr>
<td>S1</td>
<td>A4</td>
</tr>
<tr>
<td>S1</td>
<td>A5</td>
</tr>
<tr>
<td>S2</td>
<td>A6</td>
</tr>
<tr>
<td>S2</td>
<td>A7</td>
</tr>
<tr>
<td>S2</td>
<td>A8</td>
</tr>
<tr>
<td>S2</td>
<td>A9</td>
</tr>
<tr>
<td>S2</td>
<td>A10</td>
</tr>
</tbody>
</table>
Then each application will be assigned a number. The parts of the database needed to fulfil this application, called segments, are defined using the relational operators SELECT and SEMIJOIN. See Table 3.

Table 3

The Applications of the Database Expressed as the Relational Operators SELECT and SEMIJOIN.

APPLICATION 1

SELECT SALESMAN WHERE STORE# = 1

APPLICATION 2

SEMIJOIN SALESMAN WITH (SELECT STORE WHERE STORE# = 1) WHERE EXPERTISE = EQUIPMENT

APPLICATION 3

SEMIJOIN PRICE WITH (SELECT STORE WHERE STORE# = 1)

APPLICATION 4

SELECT STORE
Table 3—Continued

APPLICATION 5

SELECT PRICE WHERE MAXPRICE < $2000

SEMIJOIN SALESMAN WITH (SELECT PRICE WHERE MAXPRICE < $2000) WHERE EXPERTISE = EQUIPMENT

APPLICATION 6

SELECT SALESMAN WHERE STORE# = 2

APPLICATION 7

SEMIJOIN SALESMAN WITH (SELECT STORE WHERE STORE# = 2) WHERE EXPERTISE = EQUIPMENT

APPLICATION 8

SEMIJOIN PRICE WITH (SELECT STORE WHERE STORE# = 2)

APPLICATION 9

SELECT STORE

APPLICATION 10

SELECT PRICE WHERE MAXPRICE < $2000

SEMIJOIN SALESMAN WITH (SELECT PRICE WHERE MAXPRICE < $2000) WHERE EXPERTISE = EQUIPMENT

APPLICATION 11

SELECT SALESMAN WHERE STORE# = 3

APPLICATION 12

SEMIJOIN SALESMAN WITH (SELECT STORE WHERE STORE# = 3) WHERE EXPERTISE = EQUIPMENT

APPLICATION 13

SEMIJOIN PRICE WITH (SELECT STORE WHERE STORE# = 3)

APPLICATION 14

SELECT STORE
Algorithm

The algorithm in Figure 1 is used to define the segments of each relation.

GET SEGMENTS
INITIALIZE ALL COUNTERS TO 1
FOR I = 1 TO NUMBER OF SITES DO
   FOR J = 1 TO NUMBER OF APPLICATIONS AT SITE_i DO
      K = RELATION NUMBER OF APPLICATION(i,j)
      SEGMENT(K,COUNTER(K)) = RECORDS USED BY APPLICATION(i,j)
      COUNTER(K) = COUNTER(K) + 1
   ENDFOR
ENDFOR
END

Figure 1. An Algorithm for Getting the Segments of Each Relation Determined by the Applications of the Database.

The applications determine segments of the database. The segments consist of complete records, no vertical partitioning is done. See Table 4. If an application needs any information from a record the application is then considered to use the entire record. In other words only horizontal partitioning is done.
Table 4
The Segments of the Global Database.

<table>
<thead>
<tr>
<th>SEGMENT FOR APPLICATION 1 AT SITE 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SALESMAN</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>STORE#</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEGMENT FOR APPLICATION 2 AT SITE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALESMAN</td>
</tr>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>SMITH</td>
</tr>
<tr>
<td>SMITH</td>
</tr>
<tr>
<td>SMITH</td>
</tr>
<tr>
<td>JONES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEGMENT FOR APPLICATION 3 AT SITE 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PRICE</td>
<td></td>
</tr>
<tr>
<td>EQUIPMENT</td>
<td>MINPRICE</td>
</tr>
<tr>
<td>ATARI</td>
<td>$0000</td>
</tr>
<tr>
<td>APPLE</td>
<td>$1000</td>
</tr>
<tr>
<td>COMMODORE</td>
<td>$1000</td>
</tr>
</tbody>
</table>
### SEGMENT FOR APPLICATION 4 AT SITE 1

<table>
<thead>
<tr>
<th>STORE#</th>
<th>ADDRESS</th>
<th>EQUIPMENT</th>
<th>INVENTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>APPLE</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>ATARI</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>COMMODORE</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>APPLE</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>COMPAQ</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>IBM</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>PLAINFIELD</td>
<td>KAYPRO</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>PLAINFIELD</td>
<td>ZENITH</td>
<td>20</td>
</tr>
</tbody>
</table>

### SEGMENTS FOR APPLICATION 5 AT SITE 1

#### SALESMAN

<table>
<thead>
<tr>
<th>NAME</th>
<th>STORE#</th>
<th>EXPERTISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>1</td>
<td>APPLE</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>ATARI</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>COMMODORE</td>
</tr>
<tr>
<td>JONES</td>
<td>2</td>
<td>APPLE</td>
</tr>
</tbody>
</table>

#### PRICE

<table>
<thead>
<tr>
<th>EQUIPMENT</th>
<th>MINPRICE</th>
<th>MAXPRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATARI</td>
<td>$0000</td>
<td>$0999</td>
</tr>
<tr>
<td>APPLE</td>
<td>$1000</td>
<td>$1999</td>
</tr>
<tr>
<td>COMMODORE</td>
<td>$1000</td>
<td>$1999</td>
</tr>
</tbody>
</table>
Table 4—Continued

<table>
<thead>
<tr>
<th>Name</th>
<th>Store#</th>
<th>Expertise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>2</td>
<td>Compaq</td>
</tr>
<tr>
<td>Brown</td>
<td>2</td>
<td>IBM</td>
</tr>
<tr>
<td>Brown</td>
<td>2</td>
<td>Kaypro</td>
</tr>
<tr>
<td>Jones</td>
<td>2</td>
<td>Apple</td>
</tr>
<tr>
<td>Jones</td>
<td>2</td>
<td>Zenith</td>
</tr>
</tbody>
</table>

SEGMENT FOR APPLICATION 7 AT SITE 2

<table>
<thead>
<tr>
<th>Name</th>
<th>Store#</th>
<th>Expertise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>1</td>
<td>Apple</td>
</tr>
<tr>
<td>Brown</td>
<td>2</td>
<td>Compaq</td>
</tr>
<tr>
<td>Brown</td>
<td>2</td>
<td>IBM</td>
</tr>
<tr>
<td>Jones</td>
<td>2</td>
<td>Apple</td>
</tr>
<tr>
<td>Black</td>
<td>3</td>
<td>Compaq</td>
</tr>
<tr>
<td>Black</td>
<td>3</td>
<td>IBM</td>
</tr>
</tbody>
</table>

SEGMENT FOR APPLICATION 8 AT SITE 2

<table>
<thead>
<tr>
<th>Equipment</th>
<th>MinPrice</th>
<th>MaxPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>$1000</td>
<td>$1999</td>
</tr>
<tr>
<td>Compaq</td>
<td>$2000</td>
<td>$2999</td>
</tr>
<tr>
<td>IBM</td>
<td>$3000</td>
<td>$9999</td>
</tr>
</tbody>
</table>
Table 4--Continued

SEGMENT FOR APPLICATION 9 AT SITE 2

<table>
<thead>
<tr>
<th>STORE#</th>
<th>ADDRESS</th>
<th>EQUIPMENT</th>
<th>INVENTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>APPLE</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>ATARI</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>COMMODORE</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>APPLE</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>COMPAQ</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>IBM</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>PLAINFIELD</td>
<td>KAYPRO</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>PLAINFIELD</td>
<td>ZENITH</td>
<td>20</td>
</tr>
</tbody>
</table>

SEGMENTS FOR APPLICATION 10 AT SITE 2

SALESMAN

<table>
<thead>
<tr>
<th>NAME</th>
<th>STORE#</th>
<th>EXPERTISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>1</td>
<td>APPLE</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>ATARI</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>COMMODORE</td>
</tr>
<tr>
<td>JONES</td>
<td>2</td>
<td>APPLE</td>
</tr>
</tbody>
</table>

PRICE

<table>
<thead>
<tr>
<th>EQUIPMENT</th>
<th>MINPRICE</th>
<th>MAXPRICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATARI</td>
<td>$0000</td>
<td>$0999</td>
</tr>
<tr>
<td>APPLE</td>
<td>$1000</td>
<td>$1999</td>
</tr>
<tr>
<td>COMMODORE</td>
<td>$1000</td>
<td>$1999</td>
</tr>
</tbody>
</table>
Table 4--Continued

<table>
<thead>
<tr>
<th>SEGMENT FOR APPLICATION 11 AT SITE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SALESMAN</strong></td>
</tr>
<tr>
<td><strong>NAME</strong></td>
</tr>
<tr>
<td>BLACK</td>
</tr>
<tr>
<td>BLACK</td>
</tr>
<tr>
<td>BLACK</td>
</tr>
<tr>
<td>JOHNSON</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEGMENT FOR APPLICATION 12 AT SITE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SALESMAN</strong></td>
</tr>
<tr>
<td><strong>NAME</strong></td>
</tr>
<tr>
<td>BROWN</td>
</tr>
<tr>
<td>JONES</td>
</tr>
<tr>
<td>BLACK</td>
</tr>
<tr>
<td>JOHNSON</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SEGMENT FOR APPLICATION 13 AT SITE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PRICE</strong></td>
</tr>
<tr>
<td><strong>EQUIPMENT</strong></td>
</tr>
<tr>
<td>KAYPRO</td>
</tr>
<tr>
<td>ZENITH</td>
</tr>
<tr>
<td>STORE#</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
CHAPTER V
DEFINING FRAGMENTS

Explanation and Example

The segments may overlap, i.e., any given record may be in more than one segment. To facilitate efficient distribution of the database, the database must be partitioned into pairwise disjoint fragments. This is necessary to avoid allocating unnecessary records to the fragments for a given application. The fragments must be defined in such a way that each application uses the entire fragment or none of it. See Table 5.

Table 5
The Fragments of the Global Database.

<table>
<thead>
<tr>
<th>STORE#</th>
<th>ADDRESS</th>
<th>EQUIPMENT</th>
<th>INVENTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>APPLE</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>ATARI</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>28TH ST.</td>
<td>COMMODORE</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>APPLE</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>COMPAQ</td>
<td>20</td>
</tr>
</tbody>
</table>
Table 5—Continued

<table>
<thead>
<tr>
<th>STORE#</th>
<th>ADDRESS</th>
<th>EQUIPMENT</th>
<th>INVENTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ALPINE</td>
<td>IBM</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>PLAINFIELD</td>
<td>KAYPRO</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>PLAINFIELD</td>
<td>ZENITH</td>
<td>20</td>
</tr>
</tbody>
</table>

SALESMAN

<table>
<thead>
<tr>
<th>NAME</th>
<th>STORE#</th>
<th>EXPERTISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BROWN</td>
<td>2</td>
<td>COMPAQ</td>
</tr>
<tr>
<td>BROWN</td>
<td>2</td>
<td>IBM</td>
</tr>
<tr>
<td>BROWN</td>
<td>2</td>
<td>KAYPRO</td>
</tr>
<tr>
<td>JONES</td>
<td>2</td>
<td>ZENITH</td>
</tr>
<tr>
<td>BLACK</td>
<td>3</td>
<td>COMPAQ</td>
</tr>
<tr>
<td>BLACK</td>
<td>3</td>
<td>IBM</td>
</tr>
<tr>
<td>BLACK</td>
<td>3</td>
<td>ZENITH</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>3</td>
<td>KAYPRO</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>ATARI</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>COMMODORE</td>
</tr>
<tr>
<td>SMITH</td>
<td>1</td>
<td>APPLE</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table 5—continued

<table>
<thead>
<tr>
<th>NAME</th>
<th>STORE#</th>
<th>EXPERTISE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- F8 -</td>
<td></td>
</tr>
<tr>
<td>JONES</td>
<td>2</td>
<td>APPLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRICE</td>
</tr>
<tr>
<td></td>
<td>- F9 -</td>
<td></td>
</tr>
<tr>
<td>EQUIPMENT</td>
<td>MINPRICE</td>
<td>MAXPRICE</td>
</tr>
<tr>
<td></td>
<td>- F10 -</td>
<td></td>
</tr>
<tr>
<td>COMPAQ</td>
<td>$2000</td>
<td>$2999</td>
</tr>
<tr>
<td>IBM</td>
<td>$3000</td>
<td>$9999</td>
</tr>
<tr>
<td>KAYPRO</td>
<td>$2000</td>
<td>$2999</td>
</tr>
<tr>
<td>ZENITH</td>
<td>$2000</td>
<td>$2999</td>
</tr>
<tr>
<td></td>
<td>- F11 -</td>
<td></td>
</tr>
<tr>
<td>ATARI</td>
<td>$0000</td>
<td>$0999</td>
</tr>
<tr>
<td>COMMODORE</td>
<td>$1000</td>
<td>$1999</td>
</tr>
<tr>
<td>APPLE</td>
<td>$1000</td>
<td>$1999</td>
</tr>
</tbody>
</table>

Algorithm

To form the disjoint fragments, each pair of segments that are not disjoint form three fragments, one consisting of the intersection of the pair, the other two containing the non-intersecting parts of the pair respectively. See Figure 2.
DEFINE FRAGMENT
FOR EACH RELATION IN THE DATABASE DO
  T = NUMBER OF SEGMENTS IN THIS RELATION
  W = 1
  WHILE W < T DO
    V = W + 1
    WHILE V < T DO
      IF SEGMENT_V ∩ SEGMENT_W ≠ ∅ THEN
        T = T + 1
        SEGMENT_T = SEGMENT_W ∩ SEGMENT_V
        SEGMENT_W = SEGMENT_W - SEGMENT_T
        SEGMENT_V = SEGMENT_V - SEGMENT_T
      ENDIF
      V = V + 1
    ENDWHILE
    W = W + 1
  ENDWHILE
NEXT RELATION
END

Figure 2. An Algorithm for Splitting the Segments into Pairwise Disjoint Fragments.

After this algorithm is run the segments are disjoint. All empty segments are removed and the remaining segments are renamed as fragments.
CHAPTER VI

ALLOCATION OF FRAGMENTS

Costs

Assumptions and Simplications

After the fragments have been defined they must be allocated to the proper sites. We make the following assumptions and simplifications:

1. The byte (character) is used as the unit of space.

2. The average cost per day of a unit of space over all the sites is used as the unit space cost. This is reasonable if all the hardware is similar.

3. An average is also used for the unit retrieval cost. Each application provides, as input, both the average number of retrievals per day and the average size (in number of records) of a retrieval.

4. The average cost over all updates is used for the cost of an update. An update is considered to be applied to a single record. Multiple record updates are input as multiple updates.

5. The average is used as the cost of communication between two sites. In many cases this is a reasonable assumption even when the database is widely distributed.
The first step in this process is the allocation of fragments to all sites where the cost of owning them is less than the cost of not owning them.

We call the cost of owning a fragment CA and the cost of not owning a fragment CN.

CA is the sum of the following:
1. Cost per day of space occupied by the fragment
2. Cost per day of local retrievals
3. Cost per day of local updates
4. Cost per day of updates sent from other sites

The following formulas are used to calculate the costs:

1. The cost per day of the space occupied by a fragment is given by:
   \[(\text{cost of a space unit}) \times (\text{number of records in the fragment}) \times (\text{the size of a record})\]

2. The cost of local retrievals of a fragment is given by:
   \[(\text{average number of retrievals per day}) \times (\text{average cost of a unit retrieval}) \times (\text{average number of records in a retrieval})\]

3. The cost of local updates is given by:
   \[(\text{average cost of an update}) \times (\text{average number of updates})\]
4. The cost of updates sent from other sites is given by:

\[(\text{average number of remote updates sent per day}) \times ((\text{average cost of an update}) + (\text{average cost of communication between 2 sites}))\]

CN is the cost of retrievals of the fragment from a remote site and is given by:

\[(\text{average number of retrievals per day}) \times ((\text{average communication cost between 2 sites}) + (\text{average cost of a retrieval}) \times (\text{average number of records in a retrieval}))\]

**Algorithm**

We define the benefit of allocating a fragment to a site as \(CN - CA\). The algorithm for allocating the fragments is given in Figure 3.

```
ALLOCATE
FOR M = 1 TO NUMBER OF FRAGMENTS DO
    FOR I = 1 TO NUMBER OF SITES DO
        IF BENEFIT OF FRAGMENT_M TO SITE_I > 0 THEN
            ALLOCATE FRAGMENT_M TO SITE_I
        ENDIF
    ENDFOR
ENDFOR
END
```

Figure 3. An Algorithm for Allocating the Fragments to All Sites Where There is a Positive Benefit.
Example

Table 6 gives the statistical data used for calculating benefits and Table 7 gives an example of the benefits of the fragments in Table 5.

**Table 6**

Statistical Data for Determining the Benefit of Each Fragment to Each Site.

<table>
<thead>
<tr>
<th>COST OF A UNIT OF SPACE = .001</th>
</tr>
</thead>
<tbody>
<tr>
<td>COST OF A UNIT RETRIEVAL = .1</td>
</tr>
<tr>
<td>AVERAGE COST OF AN UPDATE = .2</td>
</tr>
</tbody>
</table>

**AVERAGE COST OF COMMUNICATION BETWEEN 2 SITES = 2**

<table>
<thead>
<tr>
<th>AVERAGE NUMBER OF RETRIEVALS PER DAY</th>
<th>AVERAGE RETRIEVAL SIZE</th>
<th>AVERAGE NUMBER OF UPDATES PER DAY</th>
<th>AVERAGE NUMBER OF REMOTE UPDATES PER DAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAGMENT 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SITE 1</td>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>SITE 2</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>SITE 3</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>FRAGMENT 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SITE 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SITE 2</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>SITE 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FRAGMENT 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SITE 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SITE 2</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SITE 3</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FRAGMENT 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SITE 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SITE 2</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SITE 3</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>FRAGMENT 5</td>
<td>AVERAGE NUMBER OF RETRIEVALS PER DAY</td>
<td>AVERAGE RETRIEVAL SIZE</td>
<td>AVERAGE NUMBER OF UPDATES PER DAY</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------</td>
<td>------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>SITE 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SITE 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SITE 3</td>
<td>8</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>FRAGMENT 6</td>
<td>SITE 1</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>SITE 2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SITE 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FRAGMENT 7</td>
<td>SITE 1</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>SITE 2</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SITE 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FRAGMENT 8</td>
<td>SITE 1</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>SITE 2</td>
<td>10</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>SITE 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FRAGMENT 9</td>
<td>SITE 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SITE 2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>SITE 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FRAGMENT 10</td>
<td>SITE 1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SITE 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SITE 3</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FRAGMENT 11</td>
<td>SITE 1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>SITE 2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>SITE 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>FRAGMENT 12</td>
<td>SITE 1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>SITE 2</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>SITE 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 7

The Benefit of Allocating the Fragments to the Sites.

<table>
<thead>
<tr>
<th>FRAGMENT</th>
<th>SITE 1</th>
<th>SITE 2</th>
<th>SITE 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.82</td>
<td>-1.18</td>
<td>-1.18</td>
</tr>
<tr>
<td>2</td>
<td>-4.47</td>
<td>13.53</td>
<td>-4.47</td>
</tr>
<tr>
<td>3</td>
<td>-4.47</td>
<td>3.53</td>
<td>3.53</td>
</tr>
<tr>
<td>4</td>
<td>-8.87</td>
<td>1.13</td>
<td>7.13</td>
</tr>
<tr>
<td>5</td>
<td>-8.87</td>
<td>-8.87</td>
<td>15.13</td>
</tr>
<tr>
<td>6</td>
<td>21.13</td>
<td>-0.87</td>
<td>-8.87</td>
</tr>
<tr>
<td>7</td>
<td>18.96</td>
<td>6.96</td>
<td>-11.04</td>
</tr>
<tr>
<td>8</td>
<td>8.96</td>
<td>14.96</td>
<td>-11.04</td>
</tr>
<tr>
<td>9</td>
<td>-4.47</td>
<td>5.53</td>
<td>-4.47</td>
</tr>
<tr>
<td>10</td>
<td>-2.27</td>
<td>-2.27</td>
<td>5.73</td>
</tr>
<tr>
<td>11</td>
<td>9.33</td>
<td>1.33</td>
<td>-6.67</td>
</tr>
<tr>
<td>12</td>
<td>4.96</td>
<td>6.96</td>
<td>-11.04</td>
</tr>
</tbody>
</table>

Note: All Benefits are Negative Where the Fragment is Not Used.
Constraints

After the fragments have been allocated according to their benefits, see Figure 4 and Tables 6 and 7, the allocation can be modified to meet reliability and storage space constraints.

<table>
<thead>
<tr>
<th>ALLOCATION OF THE FRAGMENTS TO THE SITES ACCORDING TO THEIR BENEFIT, UNCONSTRAINED BY STORAGE SPACE OR RELIABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SITE 1 2 3</td>
</tr>
<tr>
<td>FRAGMENTS</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>TOTAL SPACE</td>
</tr>
</tbody>
</table>

Figure 4. The Allocation of the Fragments to the Sites.

The problem of guaranteeing optimal global benefit (least cost) while meeting these constraints appears to be an intractable problem because all permutations on the ordering of the sites (for space) and all the subsets of the sites (for reliability) would have to be checked.
However, if the variation in the benefit of a fragment to a site is reasonably small, heuristic algorithms that check arbitrary orderings will give a benefit that is close to optimal.

**Space**

**Explanation**

All the sites are checked to see if they have sufficient storage space to store all the fragments allocated to them. If a site is found with insufficient storage space all the fragments are sorted on their benefit to this site. Starting the fragment of least benefit to this site, each fragment is removed if it is allocated somewhere else. If it is not allocated elsewhere then the sites are sorted on the benefit of allocating this fragment and, starting with the site of greatest benefit, the fragment is allocated to the first site with sufficient space and removed from the original site. This continues until the fragment is moved or all sites have been checked. This entire process is repeated until there is sufficient space to store the remaining fragments or until all fragments have been checked. If there is still insufficient space a message is issued. See Figure 5 and 6.
Algorithm

CHECK SPACE
FOR I = 1 TO NUMBER OF SITES DO
GET SPACE ALLOCATED TO SITE$_I$
GET AVAILABLE SPACE AT SITE$_I$
IF AVAILABLE SPACE < ALLOCATED SPACE THEN
SORT THE FRAGMENTS ON THEIR BENEFIT TO SITE$_I$
M = INDEX OF FRAGMENT OF LEAST BENEFIT
REPEAT
IF FRAGMENT$_M$ IS ALLOCATED ELSE WHERE THEN
REMOVE FRAGMENT$_M$ FROM SITE$_I$
ELSE
SORT THE SITES ON THE BENEFIT OF FRAGMENT$_M$
J = INDEX OF SITE OF GREATEST BENEFIT
DONE = FALSE
REPEAT
GET SPACE AVAILABLE AT SITE$_J$
IF SITE$_J$ HAS ENOUGH SPACE THEN
ALLOCATE FRAGMENT$_M$ TO SITE$_J$
REMOVE FRAGMENT$_M$ FROM SITE$_I$
DONE = TRUE
ENDIF
J = NEXT INDEX IN SORTED LIST
UNTIL DONE OR ALL SITES HAVE BEEN CHECKED
END
M = NEXT INDEX IN SORTED LIST
UNTIL AVAILABLE SPACE > ALLOCATED SPACE
OR ALL FRAGMENTS HAVE BEEN CHECKED
IF AVAILABLE SPACE < ALLOCATED SPACE THEN
PRINT "INSUFFICIENT SPACE AT SITE", I
ENDIF
ENDIF
ENDFOR
END

Figure 5. An algorithm for Removing Fragments from Sites with Insufficient Space.
### Example

**Allocation of the Fragments to the Sites Where Storage Space at Site 1 is Limited to 500 Bytes**

<table>
<thead>
<tr>
<th>SITE</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAGMENTS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>11</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL SPACE</td>
<td>252</td>
<td>852</td>
<td>288</td>
</tr>
</tbody>
</table>

Note: Fragment 1 is Moved From Site 1 to Site 2.

**Figure 6.** The Allocation of the Fragments Where Some Fragments with a Positive Calculated Benefit Are Not Allocated Due to Insufficient Storage Space.

### Reliability

**Explanation**

For each application an allowed failure rate is given. This information is read in by the fragment definition portion of the program. When the pairwise disjoint fragments are defined the fragment defined by the intersection of two segments takes the lowest allowed
failure rate of the two original segments and the other two take the allowed failure rate of the segment they are a subset of. Then each fragment has a corresponding allowed failure rate. Each site has a probability of failure that is read by the fragment allocation portion of the program. The probability of the failure of a fragment can be found by the algorithm in Figure 7.

FIND FAILURE PROBABILITY
FOR M = 1 TO NUMBER OF FRAGMENTS DO
  FAILURE RATE OF FRAGMENT_M = 1
ENDFOR
FOR I = 1 TO NUMBER OF SITES DO
  FOR M = 1 TO NUMBER OF FRAGMENTS DO
    IF FRAGMENT_M ALLOCATED TO SITE_I THEN
      FAILURE RATE OF FRAGMENT_M = FAILURE RATE OF FRAGMENT_M X FAILURE RATE AT SITE_I
    ENDFOR
  ENDFOR
ENDFOR
END

Figure 7. An Algorithm for Determining the Probability of Failure for Each Fragment.

After the fragments are allocated, according to their benefit to the sites, each fragment is checked to see if it satisfies its allowed failure rate. If it does not then the sites are sorted on the cost of owning this fragment. Starting with the site where allocation of this fragment will cost the least, the fragment is allocated to each site with enough space. This continues until the failure rate is acceptable or all sites have been checked. If the failure rate is still too high a message
stating that this fragment cannot meet its reliability specification is output.

Algorithm

The algorithm in Figure 8 is used to check the reliability.

CHECK RELIABILITY
FOR M = 1 TO NUMBER OF FRAGMENTS DO
    IF FAILURE RATE OF FRAGMENTM > ITS ALLOWED FAILURE RATE THEN
        SORT THE SITES ON FRAGMENTM'S BENEFIT
        I = INDEX OF GREATEST BENEFIT
        REPEAT
            IF FRAGMENTM NOT ALLOCATED TO SITEI AND SITEI HAS ENOUGH SPACE THEN
                ALLOCATE FRAGMENTM TO SITEI
            ENDIF
            I = NEXT INDEX IN SORTED LIST
        UNTIL FAILURE RATE OF FRAGMENTM < ALLOWED FAILURE RATE OR ALL SITES HAVE CHECKED
        IF FAILURE RATE OF FRAGMENTM > ITS ALLOWED FAILURE RATE THEN
            PRINT "RELIABILITY SPECIFICATION FOR FRAGMENT",M,"CANNOT BE MET"
        ENDF
    ENDF
ENDFOR
END

Figure 8. An Algorithm to Allocate Multiple Copies of Fragments to Meet Constraints on the Required Reliability of the Applications.

Example

Figure 9 gives an example of the allocation of the fragments constrained by the required reliability of the applications.
ALLOCATION OF THE FRAGMENTS TO THE SITES WHERE THE MAXIMUM ALLOWED FAILURE RATE FOR EACH APPLICATION IS 0.2

EXPECTED FAILURE RATES

<table>
<thead>
<tr>
<th>SITES</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.2</td>
<td>0.3</td>
<td>0.2</td>
</tr>
</tbody>
</table>

\begin{tabular}{c|ccc}
FRAGMENTS & 1 & 2 & 3 \\
\hline
1 & X &  &  \\
2 & X & X &  \\
3 &  & X & X \\
4 &  & X & X \\
5 &  &  & X \\
6 & X &  & X \\
7 & X & X &  \\
8 & X & X &  \\
9 & X & X &  \\
10 &  &  & X \\
11 & X &  & X \\
12 & X & X &  \\
\hline
TOTAL SPACE & 816 & 468 & 288 \\
\end{tabular}

Note: Fragments 2 & 9 Have Been Added to Site 1

Figure 9. The Allocation of the Fragments Where Some Fragments with a Negative Calculated Benefit Are Allocated to Meet the Reliability Constraint.

If there is no site with positive benefit of owning a particular fragment, and that fragment has a zero reliability requirement, it will not be allocated and a message is issued stating that the fragment is not allocated. See Figure 10.
CHECK ALLOCATION
  FOR M = 1 TO NUMBER OF FRAGMENTS DO
    SOMEWHERE = FALSE
    FOR I = 1 TO NUMBER OF FRAGMENTS DO
      IF FRAGMENT M ALLOCATED TO SITE I THEN
        SOMEWHERE = TRUE
        EXIT LOOP
      ENDIF
    ENDFOR
  IF NOT SOMEWHERE THEN
    PRINT "FRAGMENT",M,"NOT ALLOCATED"
  ENDFOR
ENDFOR
END

Figure 10. An Algorithm to Output a Message If Any Fragment Is Not Allocated at All.
CHAPTER VII

IMPLEMENTATION

This method has been implemented with a Pascal program.

Input and Output

The input to the program consists of three files.
1. The global database
2. Relational operators defining the applications
3. The statistical data

The first file contains all the relations in the global database. The relations are separated by a blank line.

The second file contains SELECT and SEMIJOIN relational operators. The operators can not be nested so two or more statements, using temporary relations, may be necessary to define some applications. See the section on relational operators below. When a temporary relation is defined it can be used in later applications without redefining it.

The third file contains the following information:

1. The number of relations
2. The number of records in each relation
3. The record size, in bytes, of each relation
4. The number of sites
5. The average cost of a byte of storage space
6. The average cost of a single record retrieval
7. The average cost of a single record update
8. The average cost of communication between 2 sites
9. The expected failure rate at each site
10. The storage space available at each site
11. The average daily frequency of retrieval for each application
12. The average daily frequency of update for each application
13. The maximum allowed failure rate for each application
14. The average retrieval size of each application, in number of records

See Appendix A for examples of the statistical input.

The output consists of two files. One file contains all the fragments of the database. The other gives the allocation of the fragments along with their parent relations and the records in the fragments. The output file is formatted as follows:

FRAGMENT #1
RELATION #1
RECORDS 1 2 3 4 5 6 7 8
AT SITES 3

FRAGMENT #2

: 

: 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Define Segments

Relational Operators

Only horizontal fragmentation is used. This means the records necessary for each application can be defined by the SELECT and SEMIJOIN relational operators.

Select

The SELECT operator takes the subset of the records from a relation that meet a specified boolean condition. The following is an example of the SELECT operator.

SELECT SALESMAN WHERE STORE# = 1

Semijoin

The SEMIJOIN operator takes a subset of a relation where a specified field matches a specified field in a second relation. A SEMIJOIN is actually a combination of a PROJECT operator and a JOIN operator. For example:

SEMIJOIN SALESMAN WITH TEMP WHERE EXPERTISE = EQUIPMENT

is the same as:

PROJECT (all fields in SALESMAN) FROM (SALESMAN JOIN TEMP WHERE EXPERTISE = EQUIPMENT)

The semijoin is usually useful where one of the relations has already been limited by the SELECT and/or SEMIJOIN operators this could be represented with nested statements as follows:
SEMIJOIN SALESMAN WITH (SELECT PRICE WHERE MAXPRICE < $2000) WHERE EXPERTISE = EQUIPMENT

However to simplify the programming this is done with multiple statements and a temporary relation as follows:

SELECT PRICE WHERE MAXPRICE < $2000 TO TEMP

SEMIJOIN SALESMAN WITH TEMP WHERE EXPERTISE = EQUIPMENT

Any name, other than the names of the relations in the global database, may be used for temporary relations. These temporary relations are not included in the segments used to define the disjoint fragments.

Method

A simple linear search is used for SELECT, for large files indices could be used to speed up the search. For the SEMIJOIN each relation involved is sorted on the field to be matched and a linear search is used to find the matching records.

The segment definition portion of the program reads the following data:
1. The total number of sites
2. Each segment's site number
3. The expected failure rate of each site
4. The maximum allowed failure rate of each application
5. The average number of updates and retrievals, per day, for each segment
6. The average size of retrievals for each segment
This information is used by the fragment definition portion of the program.

The segments are defined as sets of integers representing the records and then arranged according to the relations they are a subset of using the algorithm in figure 1.

Define Fragments

Method

To implement this first each relation was assigned a number and given an additional field, called NUMKEY. This field holds an integer which gives the position of the record in the relation. The records in the relations may be sorted on the primary key. This would give them an ordering so the numbers in NUMKEY would have some meaning. However, this is not necessary because the numbers are only need to be unique to a particular record. Regardless of the ordering when the numbers are assigned, each number will be associated with one particular record. With this field and the relation number each record in the database is uniquely defined by two integers. The relation number and an integer for its position in the relation. The set operations in the algorithms need only work with integers, making the program much simpler and more efficient.

After the NUMKEY field is added to the relations the segments are defined by the relational operators on the
entire records, as described above, but the segments and fragments manipulated by the program consist only of the sets of integers representing the relations and records.

Getting Data for Allocation

When the disjoint fragments are formed by the algorithm, the statistical data given for the segments, defined by the applications, must be used to assign the correct data to the fragments.

Only the fragments formed by the intersection of two segments present a problem. Each other fragment uses the data for the segment it is a subset of.

Given fragment \( T \), which is formed from the intersection of segment \( V \) at site \( I \) and segment \( W \) at site \( J \), the following is done. The maximum allowed failure rate for fragment \( T \) is the lowest of the allowed failure rates for segments \( W \) and \( V \). If \( I \) and \( J \) are different, the average retrieval size for fragment \( T \) at site \( I \) is the average retrieval size of segment \( V \) at site \( I \). A similar method is used for the allowed failure rate of fragment \( T \) at site \( J \). The average frequency of retrievals and updates and the average retrieval size are handled similarly. If \( I \) and \( J \) are the same then the frequencies of segments \( V \) and \( W \) are added and an average is used for the retrieval size. See Figure 11.
GET ALLOCATION DATA
(* SEGMENT^T IS GIVEN BY THE INTERSECTION OF SEGMENT^W
AND SEGMENT^V *)

IF ALLOWED FAILURE RATE OF SEGMENT^W >
THE ALLOWED FAILURE RATE OF SEGMENT^V THEN
ALLOWED FAILURE RATE OF SEGMENT^T =
ALLOWED FAILURE RATE OF SEGMENT^W
ELSE
ALLOWED FAILURE RATE OF SEGMENT^T =
ALLOWED FAILURE RATE OF SEGMENT^V
ENDIF

FOR I = 1 TO NUMBER OF SITES DO
  IF I IN THE SITES USING SEGMENT^V THEN
    FREQUENCY OF RETRIEVAL OF SEGMENT^T AT SITE_I =
    FREQUENCY OF RETRIEVAL OF SEGMENT^W AT SITE_I
    FREQUENCY OF UPDATE OF SEGMENT^T AT SITE_I =
    FREQUENCY OF UPDATE OF SEGMENT^W AT SITE_I
    AVERAGE RETRIEVAL SIZE OF SEGMENT^T AT SITE_I =
    AVERAGE RETRIEVAL SIZE OF SEGMENT^W AT SITE_I
    SITES OF SEGMENT^T = SITES OF SEGMENT^T + {I}
  ENDIF
  IF I IN THE SITES USING SEGMENT^W THEN
    FREQUENCY OF RETRIEVAL OF SEGMENT^T AT SITE_I =
    FREQUENCY OF RETRIEVAL OF SEGMENT^W AT SITE_I
    FREQUENCY OF UPDATE OF SEGMENT^T AT SITE_I =
    FREQUENCY OF UPDATE OF SEGMENT^W AT SITE_I
    AVERAGE RETRIEVAL SIZE OF SEGMENT^T AT SITE_I =
    AVERAGE RETRIEVAL SIZE OF SEGMENT^W AT SITE_I
    SITES OF SEGMENT^T = SITES OF SEGMENT^T + {I}
  ENDIF
  IF I IN BOTH SITES USING SEGMENT^V AND SEGMENT^W THEN
    FREQUENCY OF RETRIEVAL OF SEGMENT^T AT SITE_I =
    FREQUENCY OF RETRIEVAL OF SEGMENT^W AT SITE_I +
    FREQUENCY OF RETRIEVAL OF SEGMENT^V AT SITE_I
    FREQUENCY OF UPDATE OF SEGMENT^T AT SITE_I =
    FREQUENCY OF UPDATE OF SEGMENT^W AT SITE_I +
    FREQUENCY OF UPDATE OF SEGMENT^V AT SITE_I
    AVERAGE RETRIEVAL SIZE OF SEGMENT^T AT SITE_I =
    (AVERAGE RETRIEVAL SIZE OF SEGMENT^W AT SITE_I +
    AVERAGE RETRIEVAL SIZE OF SEGMENT^V AT SITE_I)/2
  ENDIF
ENDIF
END

Figure 11. An Algorithm to Assign Statistical Data to the Fragments.
Algorithm

After the fragments of each relation are defined by the algorithm in Figure 2, the program then checks to see that all records in the relation are covered by some fragment of that relation. See figure 12.

CHECK THAT ENTIRE RELATION IS DEFINED
number of fragments
UNION ∪ FRAGMENTt
U = 1
IF UNION ≠ THE SET OF RECORDS IN THIS RELATION THEN
PRINT(SET OF RECORDS IN THIS RELATION - UNION)
PRINT "ARE NOT DEFINED BY ANY APPLICATION"
ENDIF
END

Figure 12. An Algorithm to Check That All Records in a Relation Belong to Some Fragment.

Then the fragments of this relation are accumulated onto a single linear list of fragments covering all the relations in the database. But each fragment is labeled with the number of the relation it is a subset of.

Allocation

Benefit

In the allocation phase first the costs for
1. unit space
2. unit retrieval
3. average cost of an update
4. average communication cost
are read then, for each fragment, at each site, the frequency of updates and retrievals and the size of retrievals along with the above costs are used to compute the benefit of allocating a fragment to a site by the formulas given in the chapter on fragment allocation.

The number of remote updates issued to a site for a fragment is found by the algorithm in Figure 13.

REMOTE

NUMBER OF REMOTE UPDATES TO FRAGMENT\textsubscript{T} AT SITE\textsubscript{J} = 0
FOR K = 1 TO MAXIMUM NUMBER OF FRAGMENTS DO
  IF K IS IN THE SET OF SITES USING FRAGMENT\textsubscript{T} AND K \neq J THEN
    NUMBER OF REMOTE UPDATES TO FRAGMENT\textsubscript{T} AT SITE\textsubscript{J} =
    NUMBER OF REMOTE UPDATES TO FRAGMENT\textsubscript{T} AT SITE\textsubscript{J} +
    FREQUENCY OF UPDATES TO FRAGMENT\textsubscript{T} AT SITE\textsubscript{K}
  ENDF
ENDFOR
END

Figure 13. An Algorithm to Find the Number of Remote Updates Issued Per Day to Fragment #I at Site #J.

Space Constraints

The algorithm in Figure 5 is used for the space constraints. The subroutine GET\_SPACE, see figure 14, is used to check the space used at a site. GET\_SPACE could be run once and the space at each site stored in an array. The algorithm would run more quickly if the space at each site were stored in an array. This would, of course, require more memory to store the array.
GET SPACE(I)
    SPACE = 0
FOR L = 1 TO TOTAL NUMBER OF FRAGMENTS DO
    IF FRAGMENT_L ALLOCATED TO SITE_I THEN
        SPACE = SPACE + SIZE(L)
END

Figure 14. An Algorithm to Get the Space Used at Site #I.

Reliability Constraints

The algorithm in Figure 8 is used for the reliability constraints. Like the algorithm for the space constraints, this algorithm would run faster if the space used at each site were stored in an array.
CHAPTER VIII

COMPLEXITY

Define Segments

The algorithm for the relational operators has a complexity of \( O(r) \) for the SELECT where \( r \) is the number of records in the SELECTed relation. For the SEMIJOIN each file is sorted first. The sort can be done in time \( O(r \cdot \log_2(r)) \). Then a linear search is done so the complexity is \( O(r_1 \cdot \log_2(r_1) + r_2 \cdot \log_2(r_2)) \) where \( r_1 \) and \( r_2 \) are the number of records in the first and second relations respectively. The complexity of the algorithm GET SEGMENTS is \( O(S \cdot A \cdot M) \) where \( S \) is the number of sites, \( A \) the average number of applications at each site and \( M \) the average number of records used by each application. In the worst case \( M \) would be the average number of records in each relation. The overall complexity is therefore \( O(S \cdot A \cdot M \cdot \log(M)) \).

Define Fragments

The complexity of algorithm DEFINE_FRAG is \( O(R \cdot T^2) \) where \( R \) is the number of relations and \( T \) is the average of the number of fragments plus the number of intersections of fragments, in each relation. In the
worst case each record is a separate fragment and the complexity would be \( O(R \cdot N^2) \) where \( N \) is the average number of records in each relation.

Get Costs

The complexity of \text{GET\_COST} is \( O(F \cdot S) \) where \( F \) is the number of fragments in the global database and \( S \) is number of sites. In the worst case this would be \( O(N \cdot S) \).

Meet Space Constraints

The complexity of \text{CHECK\_SPACE} is \( O(S^3 \cdot F^2) \) in the worst case, but this would only occur if all sites had insufficient space and each fragment were allocated to one site only. It could be reduced if \( n \cdot \log(n) \) sorts were used. In the worst case \( N \) would replace \( S \).

Meet Reliability Constraints

The complexity of \text{CHECK\_RELIABILITY} is \( O(F^2 \cdot S^2) \). This could be reduced to \( O(F \cdot S^2) \) if the space at each site were stored in an array. Again this could be reduced with an \( n \cdot \log(n) \) sort. In the worst case \( N \) would replace \( S \).
CHAPTER IX.

CONCLUSION

Motzkin's (1986) method for the logical design of a distributed relational database has been presented. A new approach has been used where the cost/benefit of allocating a fragment is calculated independently for each site regardless of what is allocated to the other sites. Allocating all fragments where the benefit is positive gives an optimal compromise between locality and parallel access, and the cost of multiple updates.

Heuristics are used to meet the reliability and storage space constraints. These should retain near optimal global benefit if the variation in benefits are not extreme.

Averages are used for communication, update and retrieval costs. This method is most suitable for local area networks or any network where communication costs between any pair of sites do not vary greatly and where all the hardware is similar such as a network of IBM PC micro computers or VAX minicomputers.

This method can be implemented on a computer with reasonable time complexity and the number of input parameters that need to be estimated is small enough to be practical.
The implementation of this method has been discussed and examples of the method have been shown.
FIRST RUN

INPUT

NUMBER OF RELATIONS: 3

<table>
<thead>
<tr>
<th>RELATION</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td># OF RECORDS</td>
<td>8</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>RECORD SIZE</td>
<td>48</td>
<td>36</td>
<td>36</td>
</tr>
</tbody>
</table>

NUMBER OF SITES: 3

COSTS

SPACE UNIT: 0.001

RECORD RETRIEVAL: 0.10

RECORD UPDATE: 0.20

COMMUNICATION: 2.00

<table>
<thead>
<tr>
<th>SITE</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPECTED FAILURE RATE</td>
<td>0.20</td>
<td>0.20</td>
<td>0.20</td>
</tr>
<tr>
<td>AVAILABLE SPACE</td>
<td>900</td>
<td>900</td>
<td>900</td>
</tr>
</tbody>
</table>

NUMBER OF APPLICATIONS AT SITE 1: 6

<table>
<thead>
<tr>
<th>FREQUENCY OF RETRIEVAL</th>
<th>FREQUENCY OF UPDATE</th>
<th>MAXIMUM FAIL RATE</th>
<th>AVERAGE RETRIEVAL SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
</tbody>
</table>
**FIRST RUN INPUT -- CONTINUED**

**NUMBER OF APPLICATIONS AT SITE 2: 6**

<table>
<thead>
<tr>
<th>Frequency of Retrieval</th>
<th>Frequency of Update</th>
<th>Maximum Fail Rate</th>
<th>Average Retrieval Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.20</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
</tbody>
</table>

**NUMBER OF APPLICATIONS AT SITE 3: 4**

<table>
<thead>
<tr>
<th>Frequency of Retrieval</th>
<th>Frequency of Update</th>
<th>Maximum Fail Rate</th>
<th>Average Retrieval Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>0.20</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>5</td>
</tr>
</tbody>
</table>

**OUTPUT**

**FIRST RUN**

<table>
<thead>
<tr>
<th>Parent Relation Records Allocated to Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>Fragment +----------------------------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
SECOND RUN

INPUT

NUMBER OF RELATIONS: 3

<table>
<thead>
<tr>
<th>RELATION</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td># OF RECORDS</td>
<td>8</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>RECORD SIZE</td>
<td>48</td>
<td>36</td>
<td>36</td>
</tr>
</tbody>
</table>

NUMBER OF SITES: 3

COSTS

SPACE UNIT: 0.001

RECORD RETRIEVAL: 0.10

RECORD UPDATE: 0.20

COMMUNICATION: 2.00

<table>
<thead>
<tr>
<th>SITE</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPECTED FAILURE RATE</td>
<td>0.20</td>
<td>0.20</td>
<td>0.20</td>
</tr>
<tr>
<td>AVAILABLE SPACE</td>
<td>500</td>
<td>900</td>
<td>900</td>
</tr>
</tbody>
</table>

NUMBER OF APPLICATIONS AT SITE 1: 6

<table>
<thead>
<tr>
<th>FREQUENCY OF RETRIEVAL</th>
<th>FREQUENCY OF UPDATE</th>
<th>MAXIMUM FAIL RATE</th>
<th>AVERAGE RETRIEVAL SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
</tbody>
</table>
SECOND RUN INPUT -- CONTINUED

NUMBER OF APPLICATIONS AT SITE 2: 6

<table>
<thead>
<tr>
<th>FREQUENCY OF RETRIEVAL</th>
<th>FREQUENCY OF UPDATE</th>
<th>MAXIMUM RETRIEVAL SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.20</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.20</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0.20</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.20</td>
</tr>
</tbody>
</table>

NUMBER OF APPLICATIONS AT SITE 3: 4

<table>
<thead>
<tr>
<th>FREQUENCY OF RETRIEVAL</th>
<th>FREQUENCY OF UPDATE</th>
<th>MAXIMUM RETRIEVAL SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>0.20</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
</tr>
</tbody>
</table>

OUTPUT

SECOND RUN

<table>
<thead>
<tr>
<th>PARENT RELATION RECORDS ALLOCATED TO SITES</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>FRAGMENT +</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
THIRD RUN

INPUT

NUMBER OF RELATIONS: 3

<table>
<thead>
<tr>
<th>RELATION</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

| # OF RECORDS | 8 | 12 | 7 |
| RECORD SIZE   | 48 | 36 | 36 |

NUMBER OF SITES: 3

COSTS

SPACE UNIT: 0.001

RECORD RETRIEVAL: 0.10

RECORD UPDATE: 0.20

COMMUNICATION: 2.00

<table>
<thead>
<tr>
<th>SITE 1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

| EXPECTED FAILURE RATE | 0.20 | 0.30 | 0.20 |
| AVAILABLE SPACE       | 900  | 900  | 900  |

NUMBER OF APPLICATIONS AT SITE 1: 6

<table>
<thead>
<tr>
<th>FREQUENCY OF RETRIEVAL</th>
<th>FREQUENCY OF UPDATE</th>
<th>MAXIMUM FAIL RATE</th>
<th>AVERAGE RETRIEVAL SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.20</td>
<td></td>
</tr>
</tbody>
</table>
### THIRD RUN INPUT -- CONTINUED

**NUMBER OF APPLICATIONS AT SITE 2: 6**

<table>
<thead>
<tr>
<th>FREQUENCY OF RETRIEVAL</th>
<th>FREQUENCY OF UPDATE</th>
<th>MAXIMUM FAIL RATE</th>
<th>AVERAGE RETRIEVAL SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1/2</td>
<td>0.20</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
</tbody>
</table>

**NUMBER OF APPLICATIONS AT SITE 3: 4**

<table>
<thead>
<tr>
<th>FREQUENCY OF RETRIEVAL</th>
<th>FREQUENCY OF UPDATE</th>
<th>MAXIMUM FAIL RATE</th>
<th>AVERAGE RETRIEVAL SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>0.20</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.20</td>
<td>3</td>
</tr>
</tbody>
</table>

**OUTPUT**

**THIRD RUN**

<table>
<thead>
<tr>
<th>PARENT RELATION RECORDS ALLOCATED TO SITES</th>
</tr>
</thead>
<tbody>
<tr>
<td>------------------------------------------</td>
</tr>
<tr>
<td>FRAGMENT 1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
Appendix

Detailed Algorithm
DESIGN_DISTRIBUTED_DATABASE

CALL DEFINE_SEGMENTS
CALL DEFINE_AND_ALLOCATE_FRAGMENTS
END

(*******************************************************************)

DEFINE_SEGMENTS

CALL NUMBER_RELATIONS
CALL RELATIONAL_OPERATIONS
CALL NUMBER_SEGMENTS
END

(*******************************************************************)

(* This subroutine gives a number to each relation in *)
(* the database and adds the field NUMKEY to each *)
(* relation. Each relation, with the added numbers, is *)
(* then written to a separate. *)

(*******************************************************************)

NUMBER_RELATIONS

OPEN RELATION_FILE FOR INPUT
I=1 (* relation counter *)
WHILE NOT EOF(RELATION_FILE) DO
  READ(RELATION_FILE,RELATION_NAME)
  OUT_FILE=RELATION_NAME
  OPEN OUT_FILE FOR OUTPUT
  WRITE(OUT_FILE,I)
  WRITE(OUT_FILE,RELATION_NAME)
  WRITE(OUT_FILE,"NUMKEY")
  WHILE NOT EOLN(RELATION_FILE) DO
    READ(RELATION_FILE,FIELD_NAME)
    WRITE(OUT_FILE,FIELD_NAME)
  ENDWHILE
  K=1 (* record counter *)
  BEGIN LOOP
    READ(RELATION_FILE,FIELD_VALUE)
    IF FIELD_VALUE IS BLANK THEN EXIT LOOP
    WRITE(OUT_FILE,K)
    WRITE(OUT_FILE,FIELD_VALUE)
    K = K +1
  ENDLOOP
END WHILE

COPYRIGHT NOTICE: Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
WHILE NOT EOLN(RELATION_FILE) DO
    READ(RELATION_FILE, FIELD_VALUE)
    WRITE(OUT_FILE, FIELD_VALUE)
ENDWHILE

K=K+1
END LOOP
CLOSE(OUT_FILE)
ENDWHILE
NUMBER_OF_RELATIONS=I-1
CLOSE(RELATION_FILE)
END

(*****************************************************************)
(* This subroutine performs the SELECT and SEMIJOIN operations. *)
(* It calls the SELECT and SEMIJOIN subroutines. *)
(*****************************************************************)

RELATIONAL_OPERATORS

OPEN OPERATOR_FILE AS INPUT
OPEN SEGMENT_FILE AS OUTPUT
WHILE NOT EOF(OPERATOR_FILE) DO
    TEMPORARY=FALSE
    DONE=FALSE
    WORD=""
    READ(OPERATOR_FILE,CH)
    REPEAT
        WORD=WORD+CH
        IF EOLN(OPERATOR_FILE) THEN DONE=TRUE
        IF NOT DONE THEN READ(OPERATOR_FILE,CH)
        UNTIL CH=" " OR DONE
    IF WORD="SELECT" THEN CALL SELECT
    ELSE IF WORD="SEMIJOIN" THEN CALL SEMIJOIN
    ELSE
        PRINT(WORD,"? : SELECT OR SEMIJOIN EXPECTED")
        STOP
    ENDIF
ENDWHILE
CLOSE(OPERATOR_FILE)
CLOSE(SEGMENT_FILE)
END
SELECT
(* get relation name *)
RELATION_NAME=""
READ(OPERATOR_FILE,CH)
DONE=FALSE
REPEAT
    RELATION_NAME=RELATION_NAME+CH
    IF EOLN(OPERATOR_FILE) THEN DONE=TRUE
    (* if no conditions then done=true *)
    IF NOT DONE THEN READ(OPERATOR_FILE,CH)
UNTIL CH=" " OR DONE
RELATION_FILE="RELATION_NAME"
OPEN RELATION_FILE FOR INPUT

IF DONE THEN    (* If there are no conditions on *)
    (* the SELECT then output entire file. *)
    WHILE NOT EOF(RELATION_FILE) DO
        READ(RELATION_FILE,CH)
        WRITE(SEGMENT_FILE,CH)
    ENDWHILE
ELSE (* if not done *)

(* get keyword WHERE *)
WORD=""
READ(OPERATOR_FILE,CH)
REPEAT
    WORD=WORD+CH
    READ(OPERATOR_FILE,CH)
UNTIL CH=" ">
    IF WORD="WHERE" THEN
        PRINT (WORD," ? : WHERE EXPECTED")
        STOP
    ENDIF

(* get field name *)
FIELD=""
READ(OPERATOR_FILE,CH)
REPEAT
    FIELD=FIELD+CH
    READ(OPERATOR_FILE,CH)
UNTIL CH=" ">

(* get boolean operator *)

OPERATOR=""
READ(OPERATOR_FILE,CH)
REPEAT
    OPERATOR=OPERATOR+CH
    READ(OPERATOR_FILE,CH)
UNTIL CH=" 

(* get field value *)

READ(OPERATOR_FILE,CH)
VALUE=CH
WHILE NOT EOLN(OPERATOR_FILE) AND NOT TEMPORARY DO
    READ(OPERATOR_FILE,CH)
    IF CH=" " THEN TEMPORARY=TRUE
(* if not end of line after VALUE then TEMPORARY = TRUE *)
    VALUE=VALUE+CH
ENDWHILE

IF TEMPORARY THEN

(* get keyword TO *)

WORD=""
READ(OPERATOR_FILE,CH)
REPEAT
    WORD=WORD+CH
    READ(OPERATOR_FILE,CH)
UNTIL CH=" 
    IF WORD^"TO" THEN
        PRINT(WORD,"? :TO EXPECTED")
        STOP
    ENDIF
END

(* get temporary file name and open the file *)

READ(OPERATOR_FILE,CH)
TEMPORARY_FILE=CH
WHILE NOT EOLN(OPERATOR_FILE) DO
    READ(OPERATOR_FILE,CH)
    TEMPORARY_FILE=TEMPORARY_FILE+CH
ENDWHILE
OPEN TEMPORARY_FILE FOR OUTPUT
ENDIF

(* The statement from the operator file has now been *) (* read. *)
(* Start reading from the relation file. *)

(* get the relation number *)

READ(RELATION_FILE,RELATION_NUMBER)
IF TEMPOARAY THEN
    WRITE(TEMPORARY_FILE,RELATION_NUMBER)
ELSE
    WRITE(SEGMENT_FILE,RELATION_NUMBER)
ENDIF

(* read the relation name *)
READ(RELATION_FILE,RELATION_NAME)
IF TEMPOARAY THEN
    WRITE(TEMPORARY_FILE,RELATION_NAME)
ELSE
    WRITE(SEGMENT_FILE,RELATION_NAME)
ENDIF

(* get the field names *)
N=0
WHILE NOT EOLN(RELATION_FILE) DO
    N=N+1
    READ(RELATION_FILE,FIELD_NAME(N))
    IF TEMPOARAY THEN
        WRITE(TEMPORARY_FILE,FIELD_NAME(N))
    ELSE
        WRITE(SEGMENT_FILE,FIELD_NAME(N))
    ENDIF
ENDWHILE

(* find the SELECTED field *)
FOR I=1 TO N DO
    IF FIELD_NAME(I)=FIELD THEN
        SELECT_FIELD_POSITION=I
        EXIT LOOP
    ENDFIELD
ENDFOR

(* search the file *)
WHILE NOT EOF(RELATION_FILE) DO
    I=0
    (* find the position of the SELECTED field *)
    WHILE I<SELECT_FIELD_POSITION DO
        I=I+1
        READ(RELATION_FILE,FIELD_VALUE(I))
    ENDFIELD
    OK=FALSE
(* Find if the field value in the current record meets *)
(* the condition. *)
CASE OPERATOR OF
"=": IF FIELD_VALUE(I) = VALUE THEN OF = TRUE
"≠" : IF FIELD_VALUE(I) ≠ VALUE THEN OF = TRUE
"<": IF FIELD_VALUE(I) < VALUE THEN OF = TRUE
"<": IF FIELD_VALUE(I) < VALUE THEN OF = TRUE
">" : IF FIELD_VALUE(I) > VALUE THEN OF = TRUE
">" : IF FIELD_VALUE(I) > VALUE THEN OF = TRUE
ENDCASE

IF OK THEN (* if it does meet the condition *)

(* get the rest of the record *)
WHILE I < N DO
  I = I + 1
  READ(RELATION_FILE, FIELD_VALUE(I))
ENDWHILE

(* write the record to the output file *)
FOR I = 1 TO N DO
  IF TEMPORARY THEN
    WRITE(Temporary_FILE, FIELD_VALUE(I))
  ELSE
    WRITE(SEGMENT_FILE, FIELD_VALUE(I))
  ENDIF
ENDFOR
ENDWHILE
ENDIF
CLOSE(RELATION_FILE)
IF TEMPORARY THEN CLOSE(Temporary_FILE)
END

******************************************************************************************

SEMIJOIN

(* get the first relation name *)

RELATION_ONE = ""
READ(OPERATOR_FILE, CH)
REPEAT
  RELATION_ONE = RELATION_ONE + CH
  READ(OPERATOR_FILE, CH)
UNTIL CH = " "
(* get the key word WITH *)

    WORD=""
    READ(OPERATOR_FILE,CH)
    REPEAT
        WORD=WORD+CH
        READ(OPERATOR_FILE,CH)
    UNTIL CH=""
    IF WORD\"WITH" THEN
        PRINT(WORD,"? :WITH EXPECTED")
        STOP
    ENDIF

(* get the second relation name *)

    RELATION_TWO=""
    READ(OPERATOR_FILE,CH)
    REPEAT
        RELATION_TWO=RELATION_TWO+CH
        READ(OPERATOR_FILE,CH)
    UNTIL CH=""
    RELATION_FILE1=RELATION_ONE
    RELATION_FILE2=RELATION_TWO

(* get the keyword WHERE *)

    WORD=""
    READ(OPERATOR_FILE,CH)
    REPEAT
        WORD=WORD+CH
        READ(OPERATOR_FILE,CH)
    UNTIL CH=""
    IF WORD\"WHERE" THEN
        PRINT(WORD,"? :WHERE EXPECTED")
        STOP
    ENDIF

(* get the field name in the first relation *)

    FIELD1=""
    READ(OPERATOR_FILE,CH)
    REPEAT
        FIELD1=FIELD1+CH
        READ(OPERATOR_FILE,CH)
    UNTIL CH=""
(* get the operator, must be = *)

OPERATOR=""
READ(OPERATOR_FILE,CH)
REPEAT
  OPERATOR=OPERATOR+CH
  READ(OPERATOR_FILE,CH)
UNTIL CH=""
IF OPERATOR!="=" THEN
  PRINT(OPERATOR,"? : = EXPECTED")
  STOP
ENDIF

(* get the field in the second relation *)

READ(OPERATOR_FILE,CH)
FIELD2=CH
WHILE NOT EOLN(OPERATOR_FILE) AND NOT TEMPORARY DO
  READ(OPERATOR_FILE,CH)
  IF CH="" THEN TEMPORARY=TRUE
(* If the line does not end here a temporary file is *)
(* used. *)
  FIELD2=FIELD2+CH
ENDWHILE
IF TEMPORARY THEN

(* get the keyword TO *)

WORD=""
READ(OPERATOR_FILE,CH)
REPEAT
  WORD=WORD+CH
  READ(OPERATOR_FILE,CH)
UNTIL CH=""
IF WORD!="TO" THEN
  PRINT(WORD,"? :TO EXPECTED")
  STOP
ENDIF

(* get the temporary file name and open the file *)

READ(OPERATOR_FILE,CH)
TEMPORARY_FILE=CH
WHILE NOT EOLN(OPERATOR_FILE) DO
  READ(OPERATOR_FILE,CH)
  TEMPORARY_FILE=TEMPORARY+CH
ENDWHILE
OPEN TEMPORARY_FILE FOR OUTPUT
ENDIF
(* The statement from the operator file has now been *)
(* read. *)
(* Start reading the 2 files to be joined. *)

CLEAR ALL DATA FROM THE RELATIONS
OPEN RELATION_FILE1 FOR INPUT
OPEN RELATION_FILE2 FOR INPUT

(* get the number of the first relation *)
READ(RELATION_FILE1,RELATION_NUMBER)
IF TEMPORARY THEN
  WRITE(TMPORARY_FILE,RELATION_NUMBER)
ELSE
  WRITELN(SEGMEN_FILE,RELATION_NUMBER)
ENDIF

(* read the name of the first relation *)
READ(RELATION_FILE1,RELATION_ONE)
IF TEMPORARY THEN
  WRITE(TMPORARY_FILE,RELATION_ONE)
ELSE
  WRITELN(SEGMEN_FILE,RELATION_ONE)
ENDIF

(* get the field names of the first relation *)
S=0
WHILE NOT EOLN(RELATION_FILE1) DO
  S=S+1
  READ(RELATION_FILE1,FIELD_NAME(S))
  IF TEMPORARY THEN
    WRITE(TMPORARY_FILE,FIELD_NAME(S))
  ELSE
    WRITE(SEGMEN_FILE,FIELD_NAME(S))
  ENDIF
ENDWHILE

(* Find the position of the match field in the first *)
(* relation. *)
FOR I=1 TO S DO
  IF FIELD_NAME(I)=FIELD1 THEN
    MATCH_FIELD_POSITION_ONE=I
    EXIT LOOP
  ENDFOR
ENDFOR


(* Store relation one to be sorted. *)

I=0
WHILE NOT EOF(RELATION_FILE1) DO
  I=I+1
  J=0
  WHILE NOT EOLN(RELATION_FILE1) DO
    J=J+1
    READ(RELATION_FILE1, FIELD_ONE(I,J))
  ENDWHILE
ENDWHILE
N=I (* N = number of records in first relation *)

(* read the name of the second relation *)

READ(RELATION_FILE2, RELATION_TWO)

(* read the field names of the second relation *)
READ(RELATION_FILE2, FIELD_NAMES)

(* Find the position of the match field in the second relation *)

I=0
WHILE NOT EOLN(RELATION_FILE2) DO
  I=I+1
  READ(RELATION_FILE2, FIELD_NAME(I))
  IF FIELD_NAME(I)=FIELD2 THEN
    MATCH_FIELD_POSITION_TWO=I
    EXIT LOOP
 ENDIF
ENDIF

(* Store the second relation relation to be sorted. *)

I=0
WHILE NOT EOF(RELATION_FILE2) DO
  I=I+1
  J=0
  WHILE NOT EOLN(RELATION_FILE2) DO
    J=J+1
    READ(RELATION_FILE2, FIELD_TWO(I,J))
  ENDWHILE
ENDWHILE
M=I (* M = number of records in the second relation *)

SORT RELATION_ONE ON FIELD1
SORT RELATION_TWO ON FIELD2
(* Search the sorted files for matching fields *)
(* Output the records of the first relation when *)
(* the fields match *)

I=J=1
P1=MATCH_FIELD_POSITION_ONE
P2=MATCH_FIELD_POSITION_TWO
WHILE I<N AND J<M DO
   IF FIELD_ONE(I,P1)<FIELD_TWO(J,P2) THEN I=I+1
   IF FIELD_ONE(I,P1)>FIELD_TWO(J,P2) THEN J=J+1
   IF FIELD_ONE(I,P1)=FIELD_TWO(J,P2) THEN
      FOR K=1 TO S DO
         IF TEMPOARAY THEN
            WRITE(TEMPORARY_FILE,FIELD_ONE(I,K))
         ELSE
            WRITE(SEGMENT_FILE,FIELD_ONE(I,K))
         ENDIF
      ENDFOR
      I=I+1
   ENDIF
ENDWHILE
CLOSE(RELATI0N_FILE1)
CLOSE(RELATI0N_FILE2)
IF TEMPORARY THEN CLOSE(TEMPORARY_FILE)
END

(* This subroutine associates the segments with a *)
(* relation and give them a position in the relation. *)

NUMBER_SEGMENTS

OPEN SEGMENT_FILE FOR INPUT
OPEN DATA_FILE FOR INPUT (* the statistical data *)
READ(DATA_FILE,NUMBER_OF_RELATIONS)
FOR I=1 TO NUMBER_OF_RELATIONS DO
   READ(DATA_FILE,NUMBER_OF_RECORDS(I))
   READ(DATA_FILE,RECORD_SIZE(I))
   COUNTER(I)=1
ENDFOR
READ(DATA_FILE,NUMBER_OF_SITES)
READ(DATA_FILE,COST_OF_SPACE_UNIT)
READ(DATA_FILE,AVERAGE_COST_OF_RECORD_RETRIEVAL)
READ(DATA_FILE,AVERAGE_COST_OF_RECORD_UPDATE)
READ(DATA_FILE,AVERAGE_COMMUNICATION_COST)
FOR I=1 TO NUMBER_OF_SITES DO
   READ(DATA_FILE,FAILURE_PROBABILITY(I))
   READ(DATA_FILE,AVAILABLE_SPACE(I))
ENDFOR
CLEAR ALL SEGMENTS
FOR I = 1 TO NUMBER_OF_SITES DO
READ(NUMBER_OF_APPLICATIONS(I))
FOR J = 1 TO NUMBER_OF_APPLICATIONS(I) DO
READ(SEGMENT_FILE, RELATION_NUMBER)
READ(SEGMENT_FILE, RELATION_NAME)
READ(SEGMENT_FILE, FIELD_NAMES)
READLINE(SEGMENT_FILE, NUMKEY)
(* Only the numkey field is read, it is what will be *)
(* used by the fragment definition subroutine. *)
M = RELATION_NUMBER
N = COUNTER(RELATION_NUMBER)
SITE_OF(M, N) = I
READ(DATA_FILE, FREQUENCY_OF_RETRIEVAL(M, N))
READ(DATA_FILE, AVERAGE_SIZE_OF_RETRIEVAL(M, N))
READ(DATA_FILE, FREQUENCY_OF_UPDATE(M, N))
READ(DATA_FILE, ALLOWED_FAILURE_PROBABILITY(M, N))
WHILE NOT FINISHED WITH APPLICATION(J) DO
SEGMENT(M, N) = SEGMENT(M, N) {NUMKEY}
ENDWHILE
COUNTER(RELATION_NUMBER) = COUNTER(RELATION_NUMBER) + 1
ENDFOR
CALL OUTPUT (* Output the sets of integers to a *
(* file to be passed to the fragment *)
(* definition subroutine. *)

END

(******************************************************)
OUTPUT

OPEN PASS_FILE FOR OUTPUT
WRITE(PASS_FILE, NUMBER_OF_SITES)
WRITE(PASS_FILE, NUMBER_OF_RELATIONS)
FOR I = 1 TO NUMBER_OF_RELATIONS DO
WRITE(PASS_FILE, NUMBER_OF_RECORDS(I))
WRITE(PASS_FILE, RECORD_SIZE(I))
ENDFOR
FOR I = 1 TO NUMBER_OF_RELATIONS DO
FOR J = 1 TO NUMBER_OF_SITES DO
WRITE(PASS_FILE, SITE_OF(I, J))
WRITE(PASS_FILE, ALLOWED_FAILURE_PROBABILITY(I, J))
WRITE(PASS_FILE, FREQUENCY_OF_RETRIEVAL(I, J))
WRITE(PASS_FILE, FREQUENCY_OF_UPDATE(I, J))
WRITE(PASS_FILE, AVERAGE_SIZE_OF_RETRIEVAL(I, J))
WRITE(PASS_FILE, SEGMENT(I, J))
ENDFOR
ENDFOR
WRITE(PASS_FILE, COST_OF_SPACE_UNIT)
WRITE(PASS_FILE, AVERAGE_COST_OF_RECORD_RETRIEVAL)
WRITE(PASS_FILE,AVERAGE_COST_OF_RECORD_UPDATE)
WRITE(PASS_FILE,AVERAGE_COMMUNICATION_COST)
FOR I=1 TO NUMBER_OF_SITES DO
  WRITE(PASS_FILE,FAILURE_PROBABILITY(I))
ENDFOR
FOR I=1 TO NUMBER_OF_SITES DO
  WRITE(PASS_FILE,AVAILABLE_SPACE(I))
ENDFOR
CLOSE(PASS_FILE)
END

***********************************************************************
(* This subroutine defines and allocates the fragments. *)
***********************************************************************

DEFINE_AND_ALLOCATE_FRAGMENTS

OPEN PASS_FILE FOR INPUT
OPEN OUT_FILE FOR OUTPUT
OPEN RELATION_FILE FOR INPUT
OPEN FRAGMENT_FILE FOR OUTPUT
READ(PASS_FILE,NUMBER_OF_SITES)
CLEAR ARRAYS
TOTAL_NUMBER_OF_FRAGMENTS=0
COUNTER=1
GET NUMBER_OF_RELATIONS AND NUMBER_OF_RECORDS IN EACH
FROM PASS_FILE
FOR K=1 TO NUMBER_OF_RELATIONS DO
  GET THE SET OF ALL-RECORDS IN RELATION(K)
  CLEAR ARRAYS FOR RELATION(K)
  CALL GET_SEGMENTS(K)
  T=NUMBER_OF_SEGMENTS(K)
  WHILE W<T DO
    V=W+1
    WHILE V<T DO
      IF SEGMENT(V) \⊇ SEGMENT(W)\neq\emptyset THEN
        T=T+1
        SEGMENT(T)=SEGMENT(W) \∩ SEGMENT(V)
        SEGMENT(W)=SEGMENT(W) - SEGMENT(T)
        SEGMENT(V)=SEGMENT(V) - SEGMENT(T)
        IF ALLOWED_FAILURE_PROBABILITY_K(W) <
          ALLOWED_FAILURE_PROBABILITY_K(V) THEN
          ALLOWED_FAILURE_PROBABILITY_K(T) =
          ALLOWED_FAILURE_PROBABILITY_K(W)
        ELSE
          ALLOWED_FAILURE_PROBABILITY_K(T) =
          ALLOWED_FAILURE_PROBABILITY_K(V)
        ENDIF
      ELSE
      ENDIF
    ENDWHILE
  ENDWHILE
ENDFOR
FOR I=1 TO NUMBER_OF_SITES DO
  IF I IN SITES_OF_K(V) THEN
    FREQUENCY_OF_RETRIEVAL_K(T,I) = FREQUENCY_OF_RETRIEVAL_K(V,I)
    FREQUENCY_OF_UPDATE_K(T,I) = FREQUENCY_OF_UPDATE_K(V,I)
    AVERAGE_RETRIEVAL_SIZE_K(T,I) = AVERAGE_RETRIEVAL_SIZE_K(V,I)
    SITES_OF_K(T)=SITES_OF_K(T) {I}
  ENDIF
  IF I IN SITES_OF_K(W) THEN
    FREQUENCY_OF_RETRIEVAL_K(T,I) = FREQUENCY_OF_RETRIEVAL_K(W,I)
    FREQUENCY_OF_UPDATE_K(T,I) = FREQUENCY_OF_UPDATE_K(W,I)
    AVERAGE_RETRIEVAL_SIZE_K(T,I) = AVERAGE_RETRIEVAL_SIZE_K(W,I)
    SITES_OF_K(T)=SITES_OF_K(T) {I}
    IF I NOT IN SITES_OF_K(V) AND I IN SITES_OF_K(W) THEN
      FREQUENCY_OF_RETRIEVAL_K(T,I) = FREQUENCY_OF_RETRIEVAL_K(W,I) + FREQUENCY_OF_RETRIEVAL_K(V,I)
      FREQUENCY_OF_UPDATE_K(T,I) = FREQUENCY_OF_UPDATE_K(W,I) + FREQUENCY_OF_UPDATE_K(V,I)
      AVERAGE_RETRIEVAL_SIZE_K(T,I) = (AVERAGE_RETRIEVAL_SIZE_K(W,I) + AVERAGE_RETRIEVAL_SIZE_K(W,I))/2
      SITES_OF_K(T)=SITES_OF_K(T) {I}
    ENDIF
  ENDIF
ENDFOR

ENDIF
V=V+1
ENDWHILE
W=W+1
ENDWHILE

(* Number the fragments, eliminate any empty sets *)

U=1
FOR V=1 TO T DO
  IF SEGMENT(V)≠∅ THEN
    M=U+TOTAL_NUMBER_OF_FRAGMENTS
    FRAGMENT_K(U)=SEGMENT(V)
    ALLOWED_FAILURE_PROBABILITY_K(U) = ALLOWED_FAILURE_PROBABILITY_K(V)
    SITES_OF_K(U)=SITES_OF_K(V)
  ENDIF
ENDFOR
FOR I=1 TO NUMBER_OF_SITES DO
    FREQUENCY_OF_RETRIEVAL(M,I) = FREQUENCY_OF_RETRIEVAL_K(V,I)
    FREQUENCY_OF_UPDATE(M,I) = FREQUENCY_OF_UPDATE_K(V,I)
    AVERAGE_RETRIEVAL_SIZE(M,I) = AVERAGE_RETRIEVAL_SIZE_K(V,I)
    U=U+1
ENDFOR
ENDIF
ENDFOR

(* Determine if all records are covered by the (*)
fragments. *)

UNION=Ø
NUMBER_OF_FRAGMENTS=U-1
FOR U=1 TO NUMBER_OF_FRAGMENTS DO
    UNION=UNION J FRAGMENT_K(U)
ENDFOR
IF UNION≠RELATION(K) THEN
    WRITE(RELATION(K)-UNION)
    WRITE("ARE NOT DEFINED BY ANY APPLICATION")
ENDIF

(* Accumulate the fragments onto a single list *)

FOR M=1 TO NUMBER_OF_FRAGMENTS DO
    FRAGMENT(COUNTER)=FRAGMENT_K(M)
    REL(COUNTER)=K
    ALLOWED_FAILURE_PROBABILITY(COUNTER) =
    ALLOWED_FAILURE_PROBABILITY_K(M)
    SITES_OF(COUNTER)=SITES_OF_K(M)
    COUNTER=COUNTER+1
ENDFOR

(* Output the fragments of the Kth relation to the *) (*
fragment file *)

CALL OUTPUT_FRAGMENTS(K)

ENDFOR (* end of loop for Kth relation *)

(* Get the benefits of the fragments to the sites *)

CALL GET_COST
(* Allocate all fragments to sites where benefit is positive.*)

FOR M=1 TO TOTAL_NUMBER_OF_FRAGMENTS DO
    FOR I=1 TO NUMBER_OF_SITES DO
        IF BENEFIT(M,I) > 0 THEN
            ALLOCATE(M,I)=TRUE
        ELSE
            ALLOCATE(M,I)=FALSE
        ENDIF
    ENDFOR
ENDFOR

(* Meet space constraints *)

FOR I=1 TO NUMBER_OF_SITES DO
    GET SPACE USED AT SITE(I)
    IF AVAILABLE_SPACE(I)<SPACE USED AT SITE(I) THEN
        SORT THE FRAGMENTS ON THEIR BENEFIT TO SITE(I)
        M=HEAD
        REPEAT
            IF ELSE_WHERE(M,I) AND
                ALLOCATE(M,I)=FALSE
                THEN
            ELSE
                SORT THE SITES ON THE BENEFIT OF FRAGMENT(M)
                J=SITE_HEAD
                DONE=FALSE
                REPEAT
                    GET SPACE USED AT SITE(J)
                    IF SPACE USE AT SITE(J)+SIZE(FRAGMENT(M)) ≤ AVAILABLE_SPACE(J) THEN
                        ALLOCATE(M,J)=TRUE
                        ALLOCATE(M,I)=FALSE
                        DONE=TRUE
                    ENDFOR
                J=SITE_LINK(J)
            UNTIL DONE OR J=0
        ENDFOR
        GET SPACE USED AT SITE(I)
        M=LINK(M)
    UNTIL AVAILABLE_SPACE(I)≥SPACE USED AT SITE(I)
    OR M=0
    IF AVAILABLE_SPACE(I)<SPACE USED AT SITE(I) THEN
        WRITE(OUT_FILE,"INSUFFICIENT SPACE AT SITE",I)
    ENDFOR
ENDFOR
(* Modify allocation, if necessary, to meet reliability constraints *)

FOR M=1 TO TOTAL_NUMBER_OF_FRAGMENTS DO
    FAIL_RATE(M)=1
ENDFOR

FOR I=1 TO NUMBER_OF_SITES DO
    FOR M=1 TO TOTAL_NUMBER_OF_FRAGMENTS DO
        IF ALLOCATE(M,I) THEN
            FAIL_RATE(M)=FAIL_RATE(M) X FAILURE_PROBABILITY(I)
        ENDIF
    ENDFOR
ENDFOR

FOR M=1 TO TOTAL_NUMBER_OF_FRAGMENTS DO
    IF FAIL_RATE(M)>ALLOWED_FAILURE_PROBABILITY(M) THEN
        SORT THE SITES ON THE BENEFIT OF FRAGMENT(M)
        I=HEAD
        REPEAT
            IF NOT ALLOCATE(M,I) AND BENEFIT(M,I)<0 THEN
                ALLOCATE(M,I)=TRUE
                GET SPACE USED AT SITE(I)
                IF AVAILABLE_SPACE(I)<SPACE_USE_AT_SITE(I) THEN
                    ALLOCATE(M,I)=FALSE
                ELSE
                    FAIL_RATE(M) = FAIL_RATE(M) X FAILURE_PROBABILITY(I)
                ENDIF
            ENDIF
            I=LINK(I)
        UNTIL FAIL_RATE(M)<ALLOWED_FAILURE_PROBABILITY(M)
    OR I=0
    IF FAIL_RATE(M)>ALLOWED_FAILURE_PROBABILITY(M) THEN
        WRITE(OUT_FILE,"RELIABILITY SPECIFICATION FOR FRAGMENT",M,"CANNOT BE MET")
    ENDFOR
ENDFOR

(* WARN OF FRAGMENTS NOT ALLOCATED *)

FOR M=1 TO TOTAL_NUMBER_OF_FRAGMENTS DO
    SOMEWHERE=FALSE
    FOR I=1 TO NUMBER_OF_SITES DO
        IF ALLOCATE(M,I) THEN
            SOMEWHERE=TRUE
            EXIT LOOP
        ENDIF
    ENDFOR
ENDFOR
IF NOT SOMEWHERE THEN
    WRITE(OUT_FILE,"FRAGMENT",M,"NOT ALLOCATED")
ENDIF
ENDFOR

(**********************************************************************
OUTPUT OF ALLOCATIONS **********************************************************************)

FOR I=1 TO TOTAL_NUMBER_OF_FRAGMENTS DO
    WRITE(OUT_FILE,"FRAGMENT #",M)
    WRITE(OUT_FILE,"RELATION ",REL(M))
    WRITE(OUT_FILE,"RECORDS",FRAGMENT(M))
    WRITE(OUT_FILE,"AT SITES")
    FOR I=1 TO NUMBER_OF_SITES DO
        IF ALLOCATE(M,I) THEN
            WRITE(OUT_FILE,I)
        ENDIF
    ENDFOR
ENDFOR

(**********************************************************************)
CLOSE(RELATION_FILE)
CLOSE(FRAGMENT_FILE)
CLOSE(PASS_FILE)
CLOSE(OUT_FILE)
END

(**********************************************************************
(* This subroutine gets the sets of integers representing the segments. *)
(* **********************************************************************)

GET_SEGMENTS(K)
I=1
READ(PASS_FILE,SITE)
SITE_OF_K(I)=SITE
READ(PASS_FILE,FREQUENCY_OF_RETRIEVAL_K(I,SITE)
READ(PASS_FILE,FREQUENCY_OF_UPDATE_K(I,SITE)
READ(PASS_FILE,AVERAGE_SIZE_OF_RETRIEVAL_K(I,SITE)
WHILE NOT DONE WITH RELATION(K) DO
    SEGMENT(I)=∅
    WHILE NOT EOLN(PASS_FILE) DO
        READ(PASS_FILE,N)
        SEGMENT(I)=SEGMENT(I) {N}
    ENDFILE
READ(PASS_FILE,SITE)
I=I+1
IF NOT DONE WITH RELATION(K) THEN
    READ(PASS_FILE, SITE)
    SITE_OF_K(I)=SITE
    READ(PASS_FILE, FREQUENCY_OF_RETRIEVAL_K(I, SITE))
    READ(PASS_FILE, FREQUENCY_OF_UPDATE_K(I, SITE))
    READ(PASS_FILE, AVERAGE_SIZE_OF_RETRIEVAL_K(I, SITE))
ENDIF
ENDWHILE
NUMBER_OF_SEGMENTS(K)=I-1
END

***************************************************************************************
(* This subroutine outputs the fragments of the Kth *)
(* relation a fragment file. *)
***************************************************************************************

OUTPUT_FRAGMENTS(K)
READ(RELATION_FILE, RELATION_NAME)
READ(RELATION_FILE, FIELD_NAMES)
FOR I=1 TO NUMBER_OF_RECORDS(K) DO
    KEY(I)=I
    J=1
    REPEAT
        READ(RELATION_FILE, RECORD_CHARACTER(I, J))
        J=J+1
    UNTIL EOLN(RELATION_FILE)
    RECORD_LENGTH(I)=J-1
ENDFOR
WRITE(FRAGMENT_FILE, RELATION_NAME)
FOR I=1 TO NUMBER_OF_FRAGMENTS DO
    WRITE(FRAGMENT_FILE, FIELD_NAMES)
    CALL SELECT(I)
ENDFOR
END

(* This subroutine is used by subroutine OUT_FRAG to *)
(* select records in the Ith fragment of the Kth *)
(* relation. *)

SELECT(I)
FOR P=1 TO NUMBER_OF_RECORDS(K) DO
    IF KEY(P) IN FRAGMENT_K(I) THEN
        WRITE(FRAGMENT_FILE, RECORD(I))
    ENDIF
ENDFOR
END

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
(*********************************************************************)
(* This subroutine gets the costs and benefits of *)
(* allocating the fragments to the sites. *)
(*********************************************************************)

GET_COST

READ(PASS_FILE,COST_OF_SPACE_UNIT)
READ(PASS_FILE,AVERAGE_COST_OF_RECORD_RETRIEVAL)
READ(PASS_FILE,AVERAGE_COST_OF_RECORD_UPDATE)
READ(PASS_FILE,AVERAGE_COMMUNICATION_COST)
FOR I=1 TO NUMBER_OF_SITES DO
  READ(PASS_FILE, FAILURE_PROBABILITY(I))
ENDFOR
FOR I=1 TO NUMBER_OF_SITES DO
  READ(PASS_FILE, AVAILABLE_SPACE(I))
ENDFOR
FOR M=1 TO TOTAL_NUMBER_OF_FRAGMENTS DO
  FOR I=1 TO NUMBER_OF_SITES DO
    SPACE_COST=COST_OF_SPACE_UNIT X FRAGMENT_SIZE(M)
    RETRIEVAL_COST =
    FREQUENCY_OF_RETRIEVAL(M,I) X
    AVERAGE_COST_OF_RECORD_RETRIEVAL X
    AVERAGE_RETRIEVAL_SIZE(M,I)
    LOCAL_UPDATE_COST =
    FREQUENCY_OF_UPDATE(M,I) X
    AVERAGE_COST_OF_RECORD_UPDATE
    REMOTE_UPDATE_COST =
    REMOTE_UPDATES(M,I) X
    AVERAGE_COST_OF_RECORD_UPDATE X
    AVERAGE_COMMUNICATION_COST
    COST_OF_ALLOCATING(M,I) =
    SPACE_COST + RETRIEVAL_COST +
    LOCAL_UPDATE_COST + REMOTE_UPDATE_COST
    COST_OF_NOT_ALLOCATING =
    FREQUENCY_OF_RETRIEVAL(M,I) X
    AVERAGE_COMMUNICATION_COST
    AVERAGE_COST_OF_RECORD_RETRIEVAL
    AVERAGE_RETRIEVAL_SIZE(M,I)
    BENEFIT(M,I) = COST_OF_NOT_ALLOCATING(M,I) -
    COST_OF_ALLOCATING(M,I)
  ENDFOR
ENDFOR
END
( * This function is used by the GET_COST subroutine *)
( * find the number of remote update that will be issued *)
( * if fragment I is allocated to site J. *)

```
FUNCTION REMOTE_UPDATES(I,J)
    REMOTE_UPDATES(I,J)=0
    FOR K=1 TO MAXIMUM_NUMBER_OF_FRAGMENTS DO
        IF K IN SITES_OF(I) AND K≠J THEN
            REMOTE_UPDATES(I,J) =
                REMOTE_UPDATES(I,J) + FREQUENCY_OF_RETRIEVAL(I,K)
        ENDIF
    ENDFOR
END
```

(*******************************************************************************)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
BIBLIOGRAPHY


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


