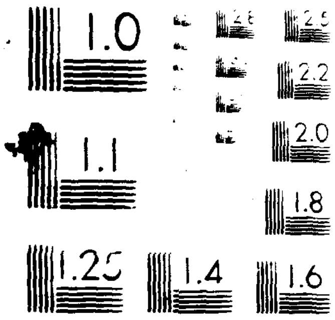
AD-A113 386 STANFORD UNIV CA DEPT OF COMPUTER SCIENCE F/G 9/2
OPTIMAL DESIGN OF DISTRIBUTED DATABASES.(U)
DEC 81 S CERI, S NAVATHE, G WIEDERHOLD N00039-80-G-0132
UNCLASSIFIED STAN-CS-81-884 NL

END
DATE
FILMED
4 82
DTIC


1.0
2.8
2.5
2.2
1.1
2.0
1.8
1.25
1.4
1.6

December 1981

Report. No. STAN-CS-81-884

12

Optimal Design of Distributed Databases

by

Stefano Ceri
Shamkant Navathe
Gio Wiederhold

Contract N00039-80-G-0132

Department of Computer Science

Stanford University
Stanford, CA 94305

NTIS

P

Stefano Ceri[†], Shamkant Navathe[‡], and Gio Wiederhold

Department of Computer Science
Stanford University, Stanford, California, 94305

Optimal Design of Distributed Databases

[†] permanent address: Istituto di Elettronica, Politecnico, Milano, Italy [‡] Dept. of Computer and Information Sciences, University of Florida, Gainesville, FL 32611

1. INTRODUCTION

The distributed information systems area has seen a rapid growth in terms of research interest as well as in terms of practical applications in the past three years. Distributed systems are becoming a reality, however truly distributed databases are still rare. For a large organization with a distributed computer network the problem of distributing a database includes determination of;

(1) How can the database be split into components to be allocated to distinct sites, and

(2) ii How much of the data should be replicated and how should the replicated fragments be allocated?

In this paper we design models for solving both of the above problems.

The problems of database distribution have been attacked earlier by researchers, but we perceive two serious shortcomings in the work known to us. First there is a body of work on file allocation which considers only a single file and ignores the complexity introduced by the interlinked files which appear in realistic databases. Second there are models which consider also the parallel problem of network topology and hence deemphasize the data distribution problem. The topology of a network with remote sites is often constrained by operational considerations, but the capabilities of network connections are such that most networks can be reconfigured to deal well with any known load.

Most modern networks provide at least on the logical level complete connectivity and have nodes that can accomodate multiple files. It is in that setting that our model is placed; we make also the simplifying assumption that the unit transmission cost is the same among any two nodes. We are then able to concentrate the problem of distribution of multi-file databases, modelled by a conceptual model of connected relations.

Figure 1 provides an outline of an overall database design methodology which is consistent with previous approaches which were proposed in a non-distributed database environment[YaNW78, LumA79]. This figure is included to define the context in which the problem is being solved. We assume that prior to undertaking the distribution of the database the following activities have been performed:

- The overall user requirements have been collected and analyzed
- Individual application views have been modelled and integrated using some formal techniques [e.g. NaSc78, WiEl80].

The specific inputs required for the "distribution design phase", obtained from the above phases, are

1. **An enterprise schema** An enterprise schema describes the global, canonical model of the information structure for the entire database. The schema may be represented by listings of relations, their attributes and domains, and definitions of connections among relations.
2. **A tabulation of transactions and their volume** The expected load of transactions to be processed using the distributed database. We assume that designers using the proposed methodology will be able to identify important transactions and give a complete specification for them. The success of the "optimization" effort is largely dependent on how accurately and completely the transactions are specified.
3. **Distribution requirements** This refers to the fact that users typically have a good understanding of how they would like to partition certain data among sites, how certain parts of data must be forced to reside at the same site, etc. These requirements are modeled as constraints in our formulation.

With the above inputs from users and designers we proceed to develop an optimization model for a non-redundant allocation of the database (Section 3). To limit the proliferation of variables we have made the following simplifications: It is assumed that all possible ways of partitioning of an object are prespecified and that the model would either select one of the candidate partitionings or allocate an object as a whole. Secondly, the logical access paths used in processing a transaction are deterministically specified. The latter allows us to focus on data distribution rather than mixing distribution with the optimization of transaction execution itself.

In spite of the above simplifications, the size of the problem for a realistic database (with tens of sites and hundreds of data entities) would still involve thousands of variables in a zero-one integer programming formulation. Since current algorithms are good only for solving problems of the order of 60 to 100 variables, it is necessary to decompose the original distribution problem into subproblems. The decomposition model is formulated as another integer program (Section 4). Finally, we develop a heuristic procedure which starts off with a given non-redundant optimal solution and determines the most beneficial replication of an object (Section 5). Section 6 includes an example of a database, a set of transactions for it, and demonstrates how the non-redundant optimization model produces different solutions for distribution as the cost parameters and frequencies of transactions are varied.

1.2 Previous Related Work

As mentioned above, the previous work has been mainly in two areas: file allocation and network topology applied to databases and communication networks.

The file allocation problem was first investigated by Chu [Chu69]. He developed a global optimization model to minimize overall operating costs under the constraints of response time and storage capacity with fixed number of copies of each file. The integer program had a very large number of variables for even small problems and was computationally infeasible. Casey [Case72] relaxed the assumption of fixed number of copies and stressed the difference between updates and retrieval. Whitney [Whit70] as well as Casey addressed the combined problem of file allocation and communication network design by restricting to tree topologies. Eswaran [Eswa74] proved that Casey's formulation was polynomially complete, hence he suggested that heuristic rather than deterministic approaches be investigated. Several studies have been made in the area of vertical partitioning and clustering of single files giving rise to integer programming or heuristic approaches [HoSc75,Ho76,HaNi79]. In the present paper we will not consider vertical partitioning of the database objects *per se* since in the distributed environment it necessitates a replication of keys and the model of transaction processing becomes too complex.

The second problem category has been explored in several studies with different sets of assumptions and addressing different sets of parameters. Mahmoud and Riordon [MaRi76] considered the combined problem of optimal file allocation and channel capacity determination, whereas Morgan and Levin [MoLe77] examined both the allocation of files and programs to process them within a generalized network. By ignoring storage capacity constraints and inroducing some other simplifying assumptions, Morgan and Levin demonstrated that the multiple file allocation problem can be decomposed into single file allocation problems. Ramamoorthy and Wah [RaWa79] analyzed a relational distributed database for optimization of query processing. By introducing redundant files, they showed how communication costs attributed to joins can be minimized. Irani and Khabbaz [IrKh79] have combined file allocation, network topology design and channel capacity allocation into a single problem. Their model minimizes the total cost of file storage and communication capacity over different channels under the constraints of a minimum level of network reliability, minimum availability of single files and maximum allowed communication delays.

2. PRELIMINARY DEFINITIONS

In order to address the general problem of distributed database design and to develop models which are widely applicable, it is necessary to define the notions of a logical database schema, occurrences of schema constructs, and the data manipulation operations in a general way. The concept of horizontal partitioning may then be applied to the individual constructs of a schema, and database transactions can be described using a small number of manipulation primitives. We summarize in this section the concepts and definitions which are necessary to develop the subsequent optimization and heuristic models. A more detailed discussion of the issues related to the modelling of logical schemas, transactions, and partitioning is presented in [NaCW81].

It is assumed that the integration of user views has already been done and that the logical schema which is subjected to distribution is a global view or an enterprise view. (See [ElWi79,LumA79] for details on views and their integration.) In our model of the logical schema, we have done away with an explicit accounting of the semantics of various relationships whenever possible, since all semantics do not have a bearing on the distribution problem. The logical schema of a database is modelled as a directed graph with objects as nodes and links as edges. Objects represent entities, events, things, or concepts of interest to a community of users. The links represent relationships among objects.

2.1 The logical schema model

We will now define the components of a logical schema, namely objects and links, which are needed for the task of designing a distribution. Included in the discussion of links is the use of join operations.

Object: An object is a BCNF relation [Codd74]. Each object has a unique primary key. A non-key column in an object typically represents an attribute of the real world object or of a relationship. An object has a unique name and index i, $1 \leq i \leq R$. An object instance is represented by an n-tuple from the object. Upper case letters $O_1, O_2, \ldots$ will denote objects, whereas lower case letters $o_1, o_2, \ldots$ will denote object instances.

Link: A link represents a binary relation among objects and specifies an ordered pair of objects. A link is described by an index h, $1 \leq h \leq L$, and may, optionally, have a name. The following functions are defined for a link l,

$$\mathit{own} : \mathcal{H} \rightarrow \mathcal{I}$$

and

$$\mathit{memb} : \mathcal{H} \rightarrow \mathcal{I},$$

where $\mathcal{H} = \{1,2,3,\ldots L\}$ and $\mathcal{I} = \{1,2,3,\ldots R\}$

These functions return the index of the owner (member) object, given the index of the link.

An instance of a link $l = (O_1, O_2)$ owned by O_1 is an ordered pair $< o_1, o_2 >$, where $o_1 \in O_1$, and $o_2 \in O_2$.

A member object instance participates in one and only one instance of a given link. In the graphic notation, the link is directed from the owner object to the member object.

One to many relationships among objects are modelled directly using a link which associates many instances of the member object with a single instance of the owner object. Many to many or "n-ary" relationships among objects ($n \geq 2$) are modeled by means of an "intersection object" which is owned by several owners via different links. Figure 2 shows some examples of the use of links.

Join Specification: Each link h has an associated join specification

$$JS_h : O_i \times O_j \rightarrow \text{Boolean, with } i = own(h), j = mem(h).$$

It maps pairs of object instances from the owner and the member of the link to true or false depending on whether or not they match the join specification. For ease of treatment, we restrict the join specification in the following discussion to the equijoin only.

Informally, JS_h is the conjunction of equipredicates of the type $O_i.C_i = O_j.C_j$, where C_i and C_j are attributes from objects O_i and O_j or columns in the corresponding relations.

We further assume that the join specification exhaustively includes those columns which constitute the primary key of the owner object.

The above idea of predefined links deserves further explanation. We recognize in the logical schema those particular equijoins which

a involve the primary key of the owner object and a compatible set of domains from the member object.

b are significant on the basis that these joins will be heavily used by transactions.

The existence of a link arises due to some real-world relationship which exists between the objects[Chen76, WiEl80]. However, in going from a high-level semantic model of the database to the logical schema for distribution design, some simplifications due to the following events are likely:

Unimportance: Relationships which express a semantic connection among objects, but are not used in any of the important transactions that are the basis for the distribution design, may be eliminated.

Unmodelled: Relationships which express a connection among objects that corresponds to a join other than the type of equijoin mentioned in (a) above, are *not* modeled as links. The set of permissible transactions however is not constrained. These joins may be performed by some transactions and give rise to an execution cost which considers the use of links whenever possible and is otherwise based on processing algorithms which do not require links.

Non-equijoins: In cases where several types of joins are possible among two objects, if a link is shown in the logical schema connecting those two objects, that link is used only for the equijoin. All other joins proceed as if no link existed.

The physical realization of a link may take several forms (*e.g.* , an index, a pointer array, etc.). If there is a link among objects `Department` and `Employee` with the join specification `Department.D#` = `Employee.D#`, it implies that given a value for `D#`, *e.g.* , `D#` = 372, it is possible to access all instances of the `Employee` object having the `D#` = 372 *without* an exhaustive search.

Quantitative parameters

Each object i has:

cardinality: total number of instances of that object, called $card(i)$.

size: total length of an object instance (tuple) in bytes, called $size(i)$.

Each link h has:

image: the fraction of the instances of the owner object participating in that link, $image(h)$.

average cardinality: the average number of instances of the member object which are associated with an owner instance which participates in the link, $avcard(h)$.

membership ratio: the ratio of the toal number of instances of the member object to the total number of instances of the owner object for the link, $ratio(h)$.

The following relationships exist:

$$ratio(h) = image(h) \times avcard(h)$$

$$card(memb(h)) = card(own(h)) \times ratio(h)$$

The cardinality of an object is shown in the logical schema by a number in the top right corner of the rectangle representing the object. The image and the average cardinality of the link are shown at the head and the tail of the arrow which represents a link (see Figure 3).

2.2 Partitioning

In this subsection we define the notion of primary and derived horizontal partitionings. These definitions are formulated with the following implicit assumptions:

Knowledge of use For a given database, it is expected that a "user" (this term is also a synonym for a "group of users" or a "team of designers" etc.) has a good understanding of the data objects and links in the logical schema and also knows the potential uses of the database at various sites. Such knowledge is often obtained during the integration phase, where the conceptual database model has been constructed [ElWi79]. This knowledge may be used in defining meaningful horizontal partitionings of data objects and the allocation of partitions to various sites.

Knowledge of linkage The user is further knowledgeable about how the partitioning applied to one object may be "propagated" to other objects via links. Propagation here implies using identical criteria for partitioning of multiple objects.

Horizontal Partitioning The horizontal partition of an object is a subdivision of the instances of the object into disjoint subsets. Each such subset, called fragment, has the same attributes as the original object, and can be allocated on a particular site of the database operational system. A horizontal partition satisfies a predicate which is a boolean expression made up of clauses:

```
<domainname><operator><value set>
```

For each object we consider two types of horizontal partitioning, resulting in primary and derived partitions.

Primary Partitioning A primary partitioning p of an object defines N_p different fragments of the object on the basis of N_p disjoint predicates.
Let $PRED(i,p)$ be the set of N predicates for partitioning the object i according to partitioning p. Let f(i,p,q) be the q^{th} fragment and let $pred_{ipq} \in PRED(i,p)$ be the predicate which defines for (i,p,q) the q^{th} fragment of object i under partitioning p. Then:

$$o \in f(i,p,q) \leftrightarrow pred_{ipq}(o)$$

$$\forall o \in O_i \exists_1 q \mid pred_{ipq}(o)$$

The allocation of fragments to the sites of the network is also given by the user. This feature, which greatly simplifies the optimization model, comes from the fact that, in real applications, the user associates candidate fragments to allocation sites in a natural way; only because of this association, the user can determine the potential convenience of a given partitioning.

The allocation of fragments is therefore an input to the application; as it will be shown later, the non-redundant optimization model determines which of the alternative candidate partitions (if any) should be applied to an object which, in turn, determines their allocation.

Derived Partitioning Derived partitioning is the concept of partitioning an object by applying the set of partitioning predicates which apply to another object so as to "derive" the partitions of the former object. Once a partitioning is defined for i, the objects which are connected via a link with this object i become candidates for derived partitioning. Two important considerations are:

appropriateness of a derived member partitioning: Given the partitioning of an owner object, it is *always* possible to define a corresponding derived partitioning of a member object via a link. Such derived partitioning may or may not be appropriate. Hence the user needs to confirm whether partionings suggested by the dependency model are to be considered.

desirability of a derived owner partitioning: Given the partitioning of a member object, it is *not necessarily* possible to define a corresponding derived partitioning of an owner object along a link. A horizontal partitioning can only be derived if object instances from any single fragment of the member object map into only one fragment of the owner object. In this case the derivation of the partitioning is feasible. In the structural model this condition is true for identity and ownership connections whenever the partitioning predicate refers to the ruling part [WiEl80].

The user must categorically state any such derived partitionings which are not only feasible but also desirable.

As a result of the above considerations a user is in a position to define "a hierarchy of derived partitionings". *E.g.*, in Figure 4, a given database has the possibility of two derived partitioning hierarchies on the basis of a primary partitioning of MED-DEPTS either by location or by a range of department numbers. These hierarchies involve a propagation of partitioning along owner-member links. The third hierarchy is based on a partitioning of PROCEDURES into surgical and non-surgical procedures. The derivation of the partitioning along PROCEDURE to MED-DEPTS and MED-DEPTS to DOCTORS proceeds from member object to owner object and is feasible since MED-DEPTS and DOCTORS can be partitioned into two corresponding subsets.

A more formal definition of derived partitioning follows:

For a given object i and its particular partitioning p, there may be an associated set $\mathcal{D}(i,p)$ of derived partitionings. An element of this set is a triple as follows:

$$\mathcal{D}(i,p) = \{< i_1, i_2, h > \mid \text{properties i, ii, iii, and iv hold}\}$$

i_1 is the object *from* which the partitioning is derived
i_2 is the object *to* which the partitioning is applied
h is the link *via* which the partitioning is derived.

i $(i_1 = own(h) \wedge i_2 = memb(h)) \bigvee ((i_2 = own(h) \wedge i_1 = memb(h))$

ii For an object instance $o \in O_{i_2}$

$$\begin{aligned} o \in fr(i_2,p,q) &\leftrightarrow pred_{i_2,p,q}(o) \\ &\leftrightarrow \exists < i_1, i_2, h > \in \mathcal{D}(i,p) \wedge \\ &\qquad o' \in O_{i_1} \mid JS_h(o,o') \wedge pred_{i_1,p,q}(o') \end{aligned}$$

iii Object i appears at least once in the first colum of $\mathcal{D}(i,p)$, and p is a primary partitioning for i

iv The second column of $\mathcal{D}(i,p)$ does not contain the same object more than once.

Note that the properties i and ii allow one to start with the object i on which the original partition is defined and define the derived partitioning for all objects separated from i by the distance of a single link. The process may be repeated to propagate a partitioning to nodes further apart than a single link.

Parameters Describing Primary Partitionings Each fragment q of partitioning p of object i has:

allocation: the site where the fragment of the database relation is potentially allocated, called $alloc(i,p,q)$

fraction: the ratio between the number of instances of the fragment and the total number of instances of the object, called [illegible](i,p,q)

Parameters Describing Derived Partitionings Each fragment q of an object i which has a derived partitioning according to partitioning p has an allocation and a fraction as before. Allocations are preserved through the join derivation; therefore it is

$$alloc(q,i,p) = alloc(q,i',p) \mid \exists\bar{i},\bar{h} \mid \bar{i},i,\bar{h} \in \mathcal{D}(i',p)$$

In absence of a more precise user specification, we assume that join predicates do not alter the fraction of primary partitions; therefore we have

$$fr(q,i,p) = fr(q,i',p) \mid \exists\bar{i},\bar{h} \mid \bar{i},i,\bar{h} \in \mathcal{D}(i',p)$$

2.3 TRANSACTION MODELING

One of the critical inputs to the distribution design process is the specification of the overall transaction load on the database. It is assumed that the users have a good notion of the important transactions that will run against the database being designed. It is expected that at least the more important transactions on the database will be identified and specified in terms of the proposed method of specification. These transactions allow an estimation of the total volume of data being accessed and transmitted. The optimization model gives a solution which minimizes the transaction processing cost.

Each transaction is described in terms of the following four basic access primitives {TDA, SDA, TJA, SJA} which are similar to the access path primitives proposed by Su et al [SuLL81].

TDA, Total Direct Access: This access primitive models the accessing of *all* instances of an object in a transaction. For an object i, the number of instances accessed is $card(i)$.

SDA, Selective Direct Access: This access primitive models the accessing of *only a selected subset* of the instances of an object in a transaction. The selection criterion is prespecified in the transaction and the *selectivity*, or the number of instances selected is *estimated* and provided by the user.

TJA, Total Join Access: This models the access *along a link*, either to the member object or to the owner object.

> **owner to member:** Access to $j = memb(h)$ from $i = own(h)$ via link h, given that N instances of i have been accessed. Then the number of instances of j accessed is $N \times ratio(h)$.
>
> **member to owner:** Access to $j = own(h)$ from $i = memb(h)$ via link L, given that N instances of i have been accessed. Then the number of instances of j accessed is N.

SJA, Selective Join Access: This models the access along a link in either direction, as in the case of TJA above. However, the number of instances of the target object accessed is *estimated*, and provided by the user. It reflects the selections made on the accessed object.

We have defined a graphic notation to go along with the above 4 access primitives, examples are shown in Figure 5.

2.3.1 Transaction Specification

The cost parameters in the optimized non-redundant model of distribution are derived from the complete specification of all important transactions known to be applied to the proposed database. Figure 6 shows a graphic description of the following transaction on the database of Figure 3.

```
FIND all Employees in the Departments in California
    who have Projects which need Part# = 7386.
LIST Emp#, Dept#, Proj#, Budget#.
```

A user interface of the type shown in Fig. 7 is postulated in which the user would essentially *trace* a transaction through the database with the numbers shown in various columns and fill in the table number of instances *touched* by the transaction wherever selective accesses are concerned. *The system* could then compute the "number accessed" by filling in the missing numbers in that column and also compute the total size of data transmitted "along the links".

2.3.2 Transaction Execution

Different distributed systems have their own ways of implementing transactions. The model of transaction execution which is implicitly assumed in the above discussion is a simple model as follows:

i A transaction originates at a specific site. The transaction has entry points corresponding to the objects which it accesses first.

ii A transaction proceeds serially by performing TDA or SDA of objects depending on whether or not a selection condition is imposed on objects. At each stage, the domains which are required to construct the end result of the transaction are forwarded to the next object.

iii A join is performed whenever a link is traversed by a transaction. A join causes the transmission of the columns which are a part of the join specification for the link concerned. The size of the join columns added to the columns required for output becomes the "size of tuple transmitted" in the specification table.

iv The "Partitioning predicate-matched" column in the transaction specification is significant to determine whether a transaction has a built-in selection predicate which matches any of the predefined horizontal partitioning predicates. The cost of the execution of a transaction reduces whenever a matched predicate is actually selected for partitioning.

v The retrieval versus update of an object has no impact on the execution of a transaction in the non-redundant distribution of data. It is, however considered important during the analysis of redundant distribution.

The transaction execution model assumes that the user has a good "understanding" of the actual execution strategy that will be employed by the system, as it requires a deterministic description of the transaction's access path. This is quite adequate for most of procedural database languages; it can, however, be critical for advanced relational systems, where the system's optimizer may behave in a different way than the user expects, and lead to unanticipated executions. In defense of the proposed approach, it can be argued that:

1 It is not now possible to combine both transaction processing optimization and database distribution optimization; because both problems are very hard and query processing optimization is highly dependent in the features of the particular system considered. Hence, when the database distribution problem alone is to be solved, a simple characterization of transaction execution cost should be given.

2 Any strategy determined by the system different than the one proposed by the user can only increase the performances of the system; hence the proposed execution strategy can be considered as a worst-case estimation.

3 The database distribution phase can be repeated once the system is operational, with better characterization of transaction execution strategies and measurements of system's loads.

2.3.3 Transaction Parameters

The following parameters are related to transactions and will be employed in the subsequent formulation. For ease of referencing they are aggregated here:

transaction index : k, $1 \leq k \leq T$
object index : i, $1 \leq i \leq R$
link index : h, $1 \leq h \leq L$

For each transaction k, the following are defined:

e_i^k = 1 if object i is the entry point of transaction
= 0 otherwise.

f_i^k = 1 if object i is the final object accessed during transaction k
= 0 otherwise.

u_i^k = 1 if object i is updated during transaction k
= 0 otherwise.

m_{ip}^k = 1 if transaction k matches partitioning predicate p for object i
= 0 otherwise.

$o(k)$ = the site of origin of transaction k.

$f(k)$ = $\{i \mid f_i^k = 1\}$ is the set of final objects in transaction k.

r_i^k = number of instances of object i selected by transaction k.

t_i^k = total number of accesses to object i made by transaction k.

Note: $t_i^k \geq r_i^k$

s_i^k = data in bytes shipped from object i to the site which stores the next object accessed by the transactions, or returned as a result when i is the final object.

$l_i^k =$ index of the link accessed next after object i.

$Next_i^k =$ index of the object accessed next after object i.

Note: $Next_i^k$ can be derived from l_i^k

3 THE NON-REDUNDANT DISTRIBUTION OF A DATABASE

In this section a model for the non-redundant distribution of a database is presented. The objective of this model is to allocate the objects to sites either wholly or in non-redundant fragments so as to minimize a global cost function. The highlights of the model are the following:

a: One of the possible alternative sites for the allocation of each object is selected.

b: The model takes into account access costs at the various sites and data transmission costs between nodes; these two criteria also ensure that the optimal solution is "good" from the viewpoint of efficiency.

c: The minimization of data transmission costs is obtained by allocating links or predefined join paths in such a way that the joins of entire objects or between partitioned objects can be performed locally.

d: The model of execution strategy for a given "logical" transaction is deterministic, and in fact the transaction consists of a sequence of retrieval or update actions on objects and of link traversal (navigation in the database) between objects. Having a non-redundant model allows evaluation of each access or transmission cost on objects or links in an additive rather than combinatorial fashion. A linear zero-one formulation is possible, involving decision variables associated with possible allocations of each object and of each link.

3.1 Variables

We define two variables, X, Y to describe the state of each object, and two variables, V, W to describe the state of each link.

$X_{ip} = 1$ if the object i is partitioned according to partition p, either in a primary or in a derived way
$= 0$ otherwise.

$Y_{ij} = 1$ if the object i is allocated on the site j as a whole
$= 0$ otherwise.

$W_{hp} = 1$ if link h is used for deriving a partition in the hierarchy of derived partitions of partitioning p, and $own(h)$ and $memb(h)$ are both partitioned using p, either in a primary or in a derived way
$= 0$ otherwise.

For a link h, there are as many W_{hp} variables as the number of distinct possible partitionings that are derived using the link.

$V_{hp} = 1$ if link h is local to site j, because $own(h)$ and $mem(h)$ are allocated on j as a whole
$= 0$ otherwise.

For a link h, there is a V_{hj} variable for every potential site j to which that link could be allocated.

3.2 Constraints

Clearly, the decision variables are constrained in order to be consistent with their definition. We have:

(1) a non-redundancy constraint

$$\sum_p X_{ip} + \sum_j Y_{ij} = 1 \qquad \text{for every object } i$$

(2) consistency constraints for variables W

$$W_{hp} \leq X_{own(h)p} \wedge W_{hp} \leq X_{mem(h)p} \quad \text{for every } W_{hp} \text{ introduced in the model}$$

(3) consistency constraints for variables V

$$V_{hj} \leq Y_{own(h)j} \wedge V_{hj} \leq Y_{mem(h)j} \qquad \text{for every } V_{hj} \text{ introduced in the model}$$

Constraints (2) and (3) are effective because the coefficients in the goal function formulation for V and W variables are negative, and hence these variables tend to assume the value 1 in the solution of the problem; however this occurs only if the X and Y variables of both the owner and member of the link to which V and W refer, *i.e.* those on the r.h.s. of the inequalities, are also set to 1.

3.3 Cost parameters

Access costs are proportional to the number of object instances accessed in a given node at a given site; further sophistication is not possible, as the physical database design at local sites will be performed in a later phase; a similar approach is taken in [TeFr80] for the logical design of a non-distributed database.

We distinguish between following types of cost units:

CLR_j	: unit cost for local retrieval accesses
CLU_j	: unit cost for local update accesses
CRR_j	: unit cost for remote retrieval accesses
CRU_j	: unit cost for remote update accesses

The above four combinations, generated by retrieval vs. update and local vs. remote access, are reasonable categories since they incur different costs attributed to authorization, concurrency, recovery, etc.

Transmission costs are proportional to the actual sizes of data involved in transmission, and not on the source and destination site. It seems impossible to give more sophisticated models as it is very difficult to predict actual costs between pairs of sites. The cost may not be fixed in some systems because of the use of dynamic routing algorithms for transmissions (*e.g.* in ARPANET); local networks(*e.g.* ETHERNET) are accurately modelled in this way.

We have:

TC: unit transmission cost between any pair of different sites j_1 and j_2;

$$TC_{j_1 j_2} > 0 \qquad \text{if} \qquad j_1 \neq j_2$$
$$TC_{j_1 j_2} > 0 \qquad \text{if} \qquad j_1 = j_2$$

3.4 Goal function

The goal function takes the form:

$$\min z = \sum_{i,p} C_{ip} X_{ip} + \sum_{i,j} D_{ij} Y_{ij} - \sum_{h,p} A_{hp} W_{hp} - \sum_{h,j} B_h V_{hj}$$

where:

C_{ip} : cost of partitioning object i according to the partition p

D_{ij} : cost of allocating the whole object i on node j

C_{ip} and D_{ij} take into account both transmission and access costs

A_{hp} : cost of transmissions which can be saved because of the use of the same partitioning criteria p on the owner and the member of the link h

B_h : cost of transmissions which are saved because both the owner and the member of the link h are stored on the same site j as a whole. Note that this parameter does not depend on a particular site j, as uniform transmission costs are assumed.

3.5 Coefficient Evaluation

The four coefficients C_{ip}, D_{ij}, A_{hp}, and B_h specify the cost of the system operation and have to be determined before the optimization of the model can take place.

3.5.1 Coefficient C_{ip} for partitioned objects

The coefficient C_{ip} gives the cost of partitioning an object i according to the partitioning p. This cost is in turn given by the sum of an access component CA_{ip} and a transmission component CT_{ip}:

$$C_{ip} = CA_{ip} + CT_{ip}$$

Both parts require some elaboration.

A) Access Component The access cost component due to partitioning is composed of the sum of costs due to all defined and relevant transactions in this partitioning:

$$CA_{ip} = \sum_{k} SCA_{ipk}$$

where SCA_{ip}, the access cost for a single transaction, is defined as:

$$\begin{aligned} SCA_{ipk} = \sum_{j \neq o(k)} NRU^{k}_{ipj} CRU_j \quad &+ \sum_{j = o(k)} NLU^{k}_{ipj} CLU_j \\ + \sum_{j \neq o(k)} NRR^{k}_{ipj} CRR_j \quad &+ \sum_{j = o(k)} NLR^{k}_{ipj} CLR_j \end{aligned}$$

The counts NRU^k_{ipj}, NLU^k_{ipj}, NRR^k_{ipj}, and NLR^k_{ipj} are respectively the number of remote updates, local updates, remote retrievals, and local retrievals for transaction k accessing the fragment on node j of the object i partitioned according to partitioning p. This cost parameter applies only to those objects which can be potentially partitioned and is evaluated separately for every candidate partitioning p.

$$\begin{aligned} NRU^k_{ipj} &= AC^k_{ipj}\, u^k_i \\ NLU^k_{ipj} &= AC^k_{ipj}\, u^k_i \\ NRR^k_{ipj} &= AC^k_{ipj}\,(1 - u^k_i) \\ NLR^k_{ipj} &= AC^k_{ipj}\,(1 - u^k_i) \end{aligned}$$

Let AC^k_{ipj} be the number of accesses of transaction k to the fragment on node j of the object i partitioned according to partition p. Recall that we defined a matching parameter m so that $m^k_{ip} = 1$ if transaction k "matches" partitioning p, i.e. , it selects objects which are potentially allocated by the partitioning on the originating site of the transaction. Then AC^k_{ipj} is defined for some q which satisfies the allocation as follows.

$$\begin{aligned} AC^k_{ipj} = \quad & \text{if } m^k_{ip} = 0 \quad \text{then } t^k_i\, fr(i,p,q) \mid alloc(i,p,q) = j \\ & \text{else if } j = o(k) \qquad \text{then} \qquad t^k_i \\ & \text{else } 0 \end{aligned}$$

i.e. if transaction k doesn't match with partition p, a uniform distribution of accesses is assumed and AC^k_{ipj} is computed as a fraction of the total accesses proportional to the fragment size; otherwise, in case of a match, two cases arise,

i: the transaction is issued on the node j that we are considering, and in this case all the accesses are made there, or

ii: the transaction is issued on a different site, and in this case we have no accesses.

B) Transmission Component The transmission component of the cost coefficient C_{ip} is given by the sum

$$CT_{ip} = CT'_{ip} + CT''_{ip}$$

where CT'_{ip} takes into account transmissions which are needed for performing the join operations which are required for performing the query or update and CT''_{ip} takes into account the transmissions of the results from the allocation sites of object i (partitioned according to p) to the site of origin of transaction $o(k)$.

The communication load for performing the joins is

$$CT'_{ip} = \sum_{\substack{q \mid \\ alloc(i,p,q) \neq o(k)}} \quad \sum_{h \mid own(h) = i} TR_{hi}\, TC$$

where TR_{hi} is the communication load per link and object fragment, specifically the number of bytes per unit time which are transmitted along link h for accessing object i.

The summation $\sum_{q|alloc(i,p,q)\neq o(k)}$ determines the number of effective fragments of object i partitioned according to p, i.e. the cardinality of the set of network sites where partitioning p places the fragments. Remember that partition predicates will match instances of object i to a subset of the network sites, and $\sum_q$ is the cardinality of such a subset. The condition $alloc(i,p,q) \neq o(k)$ is used because join information has to be transmitted to all the fragments of object i located at remote sites. TC is the cost parameter for data transmission.

The unit communication load TR_{hi} can be computed from the transaction k specifications as follows:

$$TR_{hi} = \sum_k \left(\sum_{\substack{i'|next(i')=i \wedge \\ h=l^k_{i'}}} r^k_{i'} s^k_{i'} \right)$$

i.e. , the instances $r^k_{i'}$, of the object i' which precedes i in the access path of the transaction k and therefore have to be sent to object i are weighted by their size $s^k_{i'}$; here h connects i' to i.

Note that in evaluating the transmission volumes the coefficient r^k_i is used, since it is assumed that restrictions are performed before transmitting objects in the network; in evaluating access volumes, the total access coefficients t^k_i are used instead, because instances must be actually accessed before the evaluation of restriction predicates on them.

Similarily, the cost for result transmissions is obtained by summing the communication load for the results of each transaction

$$CT''_{ip} = \sum_k SCT''_{ipk}$$

where SCT''_{ipk} is the cost for the transmission of the result from a single transaction, defined as

$$SCT''_{ipk} = \sum_{\substack{ijq|i\in f(k) \wedge \\ j\neq o(k) \wedge \\ alloc(i,p,q)=j}} (1-u^k_i)\, r^k_i\, fr(i,p,q)\, TC$$

i.e. , those fragments of the result of the transaction which are not allocated on the same site as the site of origin for the transaction, are transmitted to that site; the object i must belong to the set of terminal objects for that transaction ($i \in f(k)$), and the type of access must be retrieval.

3.5.2 Coefficient D_{ij} for whole objects

This coefficient evalutes the cost associated with objects which are wholly assigned to a particular site. It is given by the sum of an access component DA_{ij} and a transmission component DT_{ij}:

$$D_{ij} = DA_{ij} + DT_{ij}$$

These costs can again be evaluated for each transaction.

A) Access Component The access component is the sum of the access costs of single transactions SDA_{ijk}

$$DA_{ij} = \sum_k SDA_{ijk}$$

and the access cost of a single transaction SDA_{ijk} is either the sum δ of the local costs if $o(k) = \gamma$ or the remote costs.

$$\begin{aligned} SDA_{ijk} = \delta_{j,o(k)} NLU_i^k\, CLU_j \quad & + (1 - \delta_{j,o(k)}) NRU_i^k\, CRU_j \\ + \delta_{j,o(k)} NLR_i^k\, CLR_j \quad & + (1 - \delta_{j,o(k)}) NRR_i^k\, CRR_j \end{aligned}$$

The factors NLU_i^k, NRU_i^k, NLR_i^k, NRR_i^k are respectively the number of local updates, remote updates, local retrievals, and remote retrievals for transaction k accessing object i stored on any site as a whole.

$$\begin{aligned} NRU_i^k &= AC_i^k u_i^k \\ NLU_i^k &= AC_i^k u_i^k \\ NRR_i^k &= AC_i^k (1 - u_i^k) \\ NLR_i^k &= AC_i^k (1 - u_i^k) \end{aligned}$$

In this computation AC_i^k is simply the number of accesses made by transaction k to object i.

$$AC_i^k = t_i^k$$

B) Transmission Component The transmission component of the cost coefficient DT_{ij} is given similarly to the component for CT_{ij}, as the sum

$$DT_{ij} = DT'_{ij} + DT''_{ij}$$

where again DT'_{ij} takes into account transmissions which are needed for performing the join operations, and DT''_{ij} takes into account the transmission of the result, for retrieval transactions, from the node j where the object i is stored (i being one of the termination objects of the transaction's access path) to the site of origin $o(k)$.

$$DT'_{ij} = \sum_{h,j | own(h) = i} TR_{hi}\, TC$$

where TR_{hi} and TC were previously introduced; in this case, the join information has to be sent to only one site.

$$DT''_{ij} = \sum_k SDT''_{ijk}$$

where the cost of a unit transmission of the result from a single transaction SDT''_{ijk}

is defined as

$$SDT''_{ij} = \sum_{\substack{i,j \mid i \in f(k) \vee \\ j \neq o(k)}} (1 - u_i^k)\, r_i^k\, TC$$

i.e. the results of the transaction, which retrieves some information from the object i allocated on node j, have to be transmitted to the origin $o(k)$ of the transaction, if $o(k) \neq j$.

3.5.3 Coefficient A_{hp}

This coefficient evaluates the savings in performing the joins using link h when both the owner and the member objects of h are partitioned according to the same partition p; it is:

$$A_{hp} = (TR_{h\,own(h)} + TR_{h\,mem(h)})\, N_p$$

where the coefficients TR_{hi} have already been defined.

3.5.4 Coefficient B_h

This coefficient evaluates the savings in performing the joins using link h when both the owner and the member objects of h are stored as a whole on the same site j; it is:

$$B_h = TR_{h\,own(h)} + TR_{h\,mem(h)}$$

3.6 Additional Constraints for Derived Partitions

In some cases, it may be necessary to model an additional constraint. Consider a derived partition from object i' to object i'' using the link h connecting i' and i''. It could be required that a partition be induced on object i'' in the solution if and only if the same partition is also applied to the object i' in the solution. As an example, consider the case of a candidate partitioning of the **Department** object by location (for instance, **North, South, East, West**) and of a candidate derived partitioning of **Employee** objects, using the link which gives the department in which each employee works. Assume then that the candidate partitioning is selected for **Employee**, but not for **Department**. The problems which arise in this case are due to the fact that the employee information by itself is not sufficient to determine the partition and the site where the record belongs. Therefore the transaction which generates a new employee record should first join the employee record with the department record in order to derive the corresponding location, and hence determine the fragment where the record should be stored. This case is different from, for instance, the use of a partitioning criterion on the department number, which is the key field of the department object and also appears in the employee information (hence, the fragmentation criteria can be deduced without joining the corresponding relations). In conclusion, it is left to the designer to evaluate the possibility of constraining the derivation of predicates at lower levels of the derivation hierarchy to be the same as the predicate at higher levels. The constraint that models this fact is simply:

(4) Consistency constraint for derived partitions

$$X_{i''p} \leq X_{i'p} \quad \text{for constrained pairs} \; < i', i'' > \; \text{in the derivation}$$

In fact, because of this constraint, the optimization model can be simplified. The variable W_{hp} becomes useless as a result of the above constraint, because if $X_{i''p}$ is set to 1 in the solution, then certainly the link is used for deriving the partition (compare with constraints (2) in Sec. 3.2). Hence a modified model can be used in which:

a: no W_{hp} variable is introduced for those pairs of objects which are constrained in the derivation,

b: the coefficient $C_{i''p}$ is computed in the derived model as the difference of $C_{i''p}$ and A_{hp} in the original model, since the savings will always occur if the partitioning p is used for object i'',

c: the constraint (4) is introduced.

3.6.1 Modelling dependencies between objects

Similar constraints can be used for modelling other kinds of dependencies between objects. Assume that object o'' is "semantically" dependent from object o' (for instance, o'' is a weak entity in the sense of [Chen76], or or o'' is "externally" identified from o' [Nava80], or the link between o' and o'' is an ownership connection in the structural model [WiEl80]). This dependency might force the access to object o' whenever o'' is accessed, and in this case the designer could decide to force object o'' to have the same allocation of object o'. As above, we need to introduce the constraints (4'):

(4') Dependency Constraint

$$X_{i''p} \leq X_{i'p} \wedge Y_{i''j} \leq Y_{i'j} \quad \text{for dependent pairs} \; < i', i'' >$$

and the possibility of simplifying the model exists along similar lines as in the above discussion. Here both the V_{hj} and W_{hp} would be eliminated.

4 COMPUTATIONAL COMPLEXITY AND DECOMPOSITION

The computational complexity of a linear zero-one program depends roughly on the number of variables which are involved.
Let:

N : the number of sites
NP_i : the number of candidate partitions for object i
R : the number of objects
L : the number of links
ND_h : the number of partitions which are derived using the link h,

then the number of variables in the model is

$$NVAR = N(R+L) + \sum_i NP_i + \sum_h ND_h$$

(there are: NP_i variables X_{ip} and N variables Y_{ij} for each object, ND_h variables W_{ip} and N variables V_{ij} for each link)

This number can easily become too large for integer programming solution methods; hence some techniques are needed for decomposing the original design problem into subproblems which are computationally feasible. The decomposition aims at determining subsets of the enterprise schema which can be independently optimized; it is desiderable to "cut" the model into subsets by snapping the links along which the least transmission volumes occur, as the allocation of these links will not be optimized.

A model for determining such a decomposition of the problem is presented in the following; the aim of the model is to decompose the original problem into subproblems whose dimension is big enough to represent meaningful problems, yet is small enough to be computationally feasible. This aspect is captured by one of the model constraints, which limits the number of variables belonging to the subproblem between a lower and an upper bound.

4.1 Decomposition Model

The model determines a subproblem S consisting of a set of objects and links having an associated number of decision variables for the non-redundant allocation model which is limited between a lower and an upper bound. An optimal set S is determined by minimizing the volume of transmissions which use the links between objects belonging to the set and objects outside the set.

Variables One decision variable is introduced for each object i, and two decision variables are introduced for each link h.

$X_i =$ 1 if the object belongs to the subproblem S
$=$ 0 otherwise.
$Y_h =$ 1 if the link h connects an object of S and an object outside S
$=$ 0 otherwise.
$Z_h =$ 1 if the link h connects two objects of S
$=$ 0 otherwise.

Goal function The goal function for the decomposition has the form:

$$\min z = \sum_h W_h Y_h$$

where W_h represent the transmission requirements along the link h, and it is:

$$W_h = B_h = TR_{h\,own(h)} + TR_{h\,mem(h)}$$

The coefficient TR_{hi} is introduced in Sect 3.5.1.

Constraints Size constraints are introduced to assure that the subproblems created from the decomposition phase will be neither too large nor too small. There are also two new consistency constraints.

1) Constraint on the dimension of the subproblem. Let LB and UB represent the lower and upper bound respectively on the number of variables that are included in the subproblem; then:

$$LB \quad \le \quad \sum_i (N + NP_i) X_i + \sum_h (N + ND_h) Z_h \quad \le \quad UB$$

2) Consistency constraint for the variables Y_h. Each Y_h must be forced to 1 when the values assumed by the X variables of the owner and member object are different. Otherwise the Y_h value is free, and because of the positive coefficient in the goal function, Y_h will naturally be 0.

$$Y_h \ge X_{own(h)} - X_{mem(h)} \quad \wedge \quad Y_h \ge X_{mem(h)} - X_{own(h)}, \qquad 1 \le h \le L$$

3) Consistency constraints for the variables Z_h. Each Z_h must be forced to 1 when both owner and member are 1, but must be forced to 0 when any of them is 0 (hence, Z_h is equal to the product $X_{own(h)} X_{mem(h)}$). This is modelled in a linear program by introducing the constraints:

$$\begin{aligned} &Z_h \le X_{mem(h)} \\ &Z_h \le X_{own(h)} \\ &Z_h \ge X_{mem(h)} + X_{own(h)} - 1, \qquad 1 \le h \le L \end{aligned}$$

5 DESIGN OF A PARTIALLY REPLICATED DISTRIBUTION OF THE DATABASE

The introduction of redundancy in a distributed database can lead to important advantages both from the viewpoint of performance and reliability of the system. The introduction of redundancy has to be handled with great care, since it also leads to an increase in the complexity of the distributed database management software.

Redundancy exists at many level in database systems. System level redundancy is provided by data encoding and logging mechanisms, with the sole objective of gaining reliability. Redundancy in databases is commonly increased by the use of indexes and auxiliary access paths. In distributed systems especially we find that sets of data elements are kept redundantly at multiple sites.

The potential gain in performance from such a replication is due to the fact that any of the copies of each replicated database object can be used by a transaction for a retrieval access, provided that all the copies are consistent; hence several execution strategies can be used for accessing objects, decreasing overall execution costs. The improved reliability is obviously related to the availability of several copies, geographically dispersed, of the same information. The increase in the complexity of database management is mostly due to the need of maintaining the consistency of the replicated copies of the same data objects; updates have therefore to be propagated to all of them.

In the following, we will describe a heuristic technique for progressively introducing redundancy by replication, using the optimal non-redundant solution as a basis. Assumptions will be made about how transactions are handled in the replicated environment, aiming to give an execution model of transactions which is typical of distributed database systems which employ replication. It will be shown that the transaction excution model is inherently combinatorial because of replication, and this motivates the use of a heuristic allocation algorithm.

5.1 Assumptions about the Distributed Database Environment

In order to analyze replication within the distributed database environent we have to modify and extend the assumptions made in the previous analysis:

1: The non-redundancy constraint is relaxed; therefore, it is possible to have several different allocations of the same data object.

2: The updates are immediately propagated to all the copies of each data object; therefore updates to objects are directed to all the sites where the objects are replicated.

3: The optimizer which determines the execution strategy of transactions has the following features:

a: It has a global knowledge of database; global directories are therefore available at each site where transactions are optimized.

b: It selects the best alternative among the logically equivalent execution strategies which are possible for retrieval accesses; this choice reflects the same criteria which are used in the optimization model, namely, the minimization of access and transmission costs.

4: The replicated copies are used to enhance system reliability as well, and lessen the requirements for stable storage at each node [MiWi81]. The increase in reliability which depends on the presence of multiple copies can be taken into account by associating to each object a set of negative cost parameters, each of which estimates the overall benefit which is a function of the number of copies of the object.

5.2 Example of Transaction Execution over a Redundantly Distributed Database

Consider a simple transaction T^k which accesses two objects O_1 and O_2 which have degrees of redundancy dr_1 and dr_2 respectively. Consider the following 4 possible cases:

case a: Both O_1 and O_2 are retrieved ($u_1^k = u_2^k = 0$); then there are $dr_1 \times dr_2$ possible execution strategies for the transaction, each using one particular pair of copies of O_1 and O_2. The cost associated with transaction execution is the minimum cost among these alternatives.

case b: O_1 is retrieved and O_2 is updated ($u_1^k = 0$, $u_2^k = 1$); then all the copies of O_2 are accessed, while there are dr_1 alternative execution strategies for accessing one of the copies of O_1. The information which is used for joining O_1 and O_2 is sent from the selected copy of O_1 to all copies of O_2. The cost asociated with transaction execution is the minimum cost among the dr_1 alternatives.

case c: O_1 is updated and O_2 is retrieved ($u_1^k = 1$, $u_2^k = 0$); then all the copies of O_1 must be accessed, but only one copy of O_2 is sufficient. The information which is used for joining O_1 and O_2 is sent from the most convenient copy of O_1 to the selected copy of O_2. Again, the cost associated with transaction execution is the minimum among dr_2 alternatives.

case d: Both O_1 and O_2 are updated ($u_1^k = u_2^k = 1$); then there is only one execution strategy for T^k, consisting of accessing all the copies of both O_1 and O_2. For each object of O_2, the information which is used for joining O_1 and O_2 is sent from the most convenient copy of O_1.

In the following, some definitions are given which are useful for the replicated optimization model.

Let S be a non-redundant solution, O be the set of objects of the database, I be a subset of O. The set $SOL(S, I)$ contains those solutions S' generated by taking all possible ways in which objects in I can be non-redundantly allocated in combination with replicated allocations for the objects in $O-I$. Therefore, indicating with prime the allocation variables of S' and without prime the allocation variables of S, the following definition can be given:

$$\begin{aligned}
SOL(S,I) = \{S' \mid\ & \\
& \forall i \in I, ((\exists_1 p \mid X_{ip} = X'_{ip} = 1 \wedge (\forall p' \neq p, X'_{ip'} = 0) \wedge (\forall j, Y'_{ij} = 0)) \\
& \quad \vee (\exists_1 j \mid Y_{ij} = Y'_{ij} = 1 \wedge (\forall j' \neq j, Y'_{ij'} = 0) \wedge (\forall p, X'_{ip} = 0))), \\
& \forall i \in (O - I), (X'_{ip} = X_{ip} \wedge Y'_{ij} = Y_{ij})\}
\end{aligned}$$

The cardinality of $SOL(S,I)$ is a function of the degree of redundancy dr_i of the objects in I; it is $|SOL(S,I)| = \prod_{i \in I} dr_i$.

Finally, we can say that two solutions S' and S'' differ in one variable V, or $S' - S'' = V$, when $X'_{ip} = X''_{ip}$ and $Y'_{ij} = Y''_{ij}$ for all variables other than V, and $V' = 1$, $V'' = 0$.

Given a solution S, it is possible to evaluate the transaction execution cost $C(T^k, S)$ as the minimum of the set of alternative transaction execution costs $TEC(T^k, S')$, where S' is one of the solutions in $SOL(S, I^k)$ and I^k is the set of the objects retrieved by the transaction. We have:

$$C(T^k, S) = \min_{S' \in SOL(S, I^k)} TEC(T^k, S')$$

Recalling the cases of Sect. 5.2, the set of retrieved objects is

$$\begin{array}{rll}
 & I^k = \{O_1, O_2\} & \text{in case a,} \\
 & I^k = \{O_1\} & \text{in case b,} \\
 & I^k = \{O_2\} & \text{in case c,} \\
 & I^k = \Phi & \text{in case d;} \\
\text{in general it is} & I^k = \{O_i \mid u_i^k = 0\}. &
\end{array}$$

Figure 8 shows the for the 4 cases of Sect. 5.2 the accesses of transactions and the required transmissions in terms of the set of $SOL(S, I)$, given a solution S.

5.3 Description of the Object Allocation for the Redundant Database Distribution Algorithm

The description of a redundant database allocation uses the same variables X_{ip} and Y_{ij} that were introduced in the non-redundant model, releasing the non-redundancy constraint. Therefore it is possible now to allocate an object according to several alternative partitionings, or to store it as a whole on several partitionings and full allocations. We can define a redundant solution S as the assignment of 0/1 values to the decision variables X_{ip} and Y_{ij}, subjected to the constraint that each object should be allocated at least once in the distributed database; the non-redundancy constraint is therefore modified as follows:

$$\sum_p X_{ip} + \sum_j Y_{ij} \geq 1, \qquad \forall i \leq R$$

5.4 Evaluation of Transaction Execution Cost

In this section the transaction execution cost $TEC(T^k, S')$ for a given solution S' of $SOL(S, I^k)$ is derived from the logical description of transaction accesses, introduced in Section 2.3. The same cost parameters as in the non-redundant model are used for access and transmission costs (see Section 3.3).

The transaction execution cost $TEC(T^k, S')$ comprises 2 components: access costs $AC(T^k, S')$ and transmission costs $TC(T^k, S')$.

Access Cost Let $U(k)$ be the set of objects used by transaction k. The access cost $AC(T^k, S')$ considers essentially the same components as in the non-redundant model, but now aggregation is made on the objects for a fixed transaction instead of aggregating on transactions while keeping the object fixed. It is:

$$AC(T^k, S') = \sum_{i \in U(k)} (SCA_{ipk} X_{ip} + SDA_{ijk} Y_{ij})$$

where SCA_{ipk} and SDA_{ijk} were defined in Secs. 3.5.1 and 3.5.2. However, X_{ip} and Y_{ij} are now fixed and appear in the cost evaluation. Also, notice that there can be more than one value X_{ip} or Y_{ij} set to 1 for the objects which are updated.

Transmission Cost The transmission cost $TC(T^k, S')$ is given, as before, by the sum of two components. The first one, $TC'(T^k, S')$, takes into account those transmissions which are required for performing the joins. For every copy of object i which is accessed via a join with the object i' (*i.e.* for every pair $<i, i'>$ such that i' precedes i in the access path of the transaction), a transmission is required, unless i' has the same allocation as i. Therefore,

a: the transmission of the join information from object i' to all the fragments of object i is required when the solution S' has the variable X_{ip} set to 1 and $X_{i'p}$ set to 0;

b: likewise, the transmission of the join information from object i' to object i is required when the solution S' has the variable Y_{ij} set to 1 and $Y_{i'j}$ set to 0.

We have:

$$TC'(T^k, S') = \sum_{\substack{i,i',p \mid i \in U(k) \wedge \\ Next^k(i') = i}} (1 - X_{i'p})\, X_{ip}\, N_p\, r_i^k\, s_i^k + $$

$$\sum_{\substack{i,i',j \mid i \in U(k) \wedge \\ Next^k(i') = i}} (1 - Y_{i'j})\, Y_{ij}\, r_i^k\, s_i^k\, TC$$

This formulation takes care of the minimization of the transmission cost when i' and i are allocated according to the same partitioning, or on the same site; otherwise transmission of join information to each copy which is retrieved or updated is provided. The minimization of transaction execution costs which accrue to the choice of one particular copy of each retrieved object is part of the minimization of costs associated with alternative solutions in $SOL(S, I^k)$.

Note the similarity between $TC'(T^k, S')$ and the coefficients CT'_{ip} and DT'_{ij} of the non-redundant model; here again X_{ip} and Y_{ij} are fixed, and hence the use of their product in the formulation is possible.

The second component of the transmission cost, $TC''(T^k, S')$, takes into account these transmissions which are required for collecting the result of the transaction on the site of the transaction. This cost is evaluated exactly in the same way as in the non-redundant model origin, as it involves transmissions to the origin site from those sites which store terminal objects of the transaction; note that, as terminal objects are accessed for retrieval, they are not redundantly allocated. We have:

$$TC''(T^k, S') = \sum_k (SCT''_{ipk} X_{ip} + SDT''_{ijk} Y_{ij}) TC$$

where SCT''_{ipk} and SDA''_{ijk} were introduced in Secs. 3.51 and 3.52.

5.5 A Greedy Heuristic Algorithm for the Progressive Introduction of Redundancy

A greedy heuristic algorithm for the progressive introduction of redundancy, using the optimal non-redundant solution as a basis, is shown in Fig. 9. The algorithm has the following features:

1 The algorithm is iterative; at each iteration, the solution S determined at the previous iteration is taken as a basis, and all variables V which have value 0 in that solution are tentatively set to 1, generating a set of alternatives solutions S' such that $S' - S = V$. Global costs are then evaluated for all alternatives, and the one with minimal cost is selected. Therefore, at each step the "degree of redundancy" of the solution increases. The optimal solution from the non-redundant optimization model is the basis for the first iteration.

2 The algorithm can be classified as a greedy heuristic, because at each step the variable V is selected which decreases the overall costs the most.

3 The algorithm is convergent toward a relative minimum, as the overall cost monotonically decreases with progressively determined solutions. In fact, the algorithm terminates when it is not possible to decrease the overall cost any further.

4 The reliability benefit accrued by having multiple copies is not attributed to any particular transaction, but rather to a solution.

The algorithm compares alternative solutions S by associating to them a global cost $C(S)$ which is based on the cost of transaction execution and the benefits accruing from the increasing reliability due to the introduction of redundancy.

The object i whose allocation variable value is changed from 0 to 1 in S' divides the transactions into two sets. The first one consists of those transactions which use $O_i(i \in U(k))$, whose execution cost has to be evaluated. The second one consists of those transactions which do not use $O_i(i \not\in U(k))$, whose execution costs does not change from the previous iteration. Clearly, by storing individual transaction execution costs corresponding to the current solution at each iteration, these costs need not be evaluated for transactions of the second set. The transaction execution component of the global cost C' is evaluated by summing the contributions from the transactions of both sets (see Fig. 9).

The reliability benefit can be modeled as a function of dr_i, the number of copies of each object i in the considered solution. Realistically, the benefit increases with dr_i in a non-linear way; in fact, while the introduction of the first copy is highly beneficial, the interest in having the $(dr_i + 1^{th})$ copy of the same information decreases with dr_i. A possible function to model this property is:

$$f(dr_i) = (1 - 2^{-dr_i+1})B_i$$

where B_i is the benefit of having the object i infinitely redundant; note that $f(1) = 0$, $f(2) = 1/2\,B_i$, $f(3) = 3/4\,B_i$, and so on. The benefits due to replication are taken into account by summing the values returned by the above functions for all objects in S' (see Fig. 9).

At each iteration, S_{new} corresponds to the current "best" candidate between the alternative S' solutions generated from S; C, C' and C_{new} denote the corresponding global costs. The algorithm terminates when none of the candidate solutions S' yields a global cost which is less than the current global cost, and therefore $C_{new} < C$ is not satisfied.

5.6 Alternative Heuristic Formulations

The heuristic proposed above is "conservative", in the sense that decisions taken at each step involve the repetition of all cost evaluations concerning those objects whose allocation is modified from the previous step. In that sense the proposed heuristic is "sound" (decisions are always based on correct evaluations), but it is also rather hard from a computational viewpoint. As already shown, the computation of transaction execution costs grow combinatorially with the number of objects retrieved by the transaction itself; therefore the complexity of an iteration decreases linearly as the algorithm evolves because of the reduction of variables to be considered as candidates, but the complexity of transaction cost evaluation increases combinatorially with the degree of redundancy of objects in the base solution.

In some cases, the dimensions of the database design problem are such that the computational complexity of the proposed heuristic is too hard; then the proposed heuristic is a good basis for building "faster" heuristics, which sacrifice the accuracy of the final result in order to avoid hard computations. In the following, such a faster heuristic is presented which has the important property of being convergent toward a relative minimum.

The algorithm consists of the following steps:

step 1 Iteration 1 of the original algorithm is performed, and all the candidate solutions S' corresponging to a cost C' which improve the cost C^* of the optimal non-redundant solution are retained; these solutions are ranked in descending order by value of the associated cost C' (hence, the first solution correspond to the more convenient variable to be set at 1).

step 2 Then, variables are tentatively set to 1 according to the ranking order, thus increasing progressively the degree of redundancy of the solution. At each iteration, it is verified that the global cost has decreased with respect to the previous iteration; however, this is done for the considered variable only.

step 3 If the above verification fails, then it is possible either to terminate the process or to repeat Step 1 with the current solution as a basis, rank variables according to their convenience, and then proceed with Step 2. In this case, the optimization process terminates when Step 1 is performed without finding any new solution which decreases the cost of the current one.

5.7 Introduction of Concurrency Control Costs in the Design

Synchronization costs can be introduced in the design of the redundant distribution of a database to take into account the increase in complexity of concurrency control due to the presence of redundant copies, which require updating within transactions. An additional parameter CM is introduced which measures the cost of transmitting one message between different sites. Concurrency control is also necessary for databases without replication and for auditable retrieval transactions as well as for update operations. The coefficient $CC(T^k, S')$ will evalute the amount of overhead with respect to a non-redundant execution of the same transaction. Since our model aims to be general no particular schema is assumed for the implementation of the concurrency protocol, and we consider that synchronization overhead is proportional with the number of copies to be updated. This number is furthermore proportional to the number of fragments per partitioned object N_p, when the transaction does not match the partition. We have therefore

$$CC(T^k, S') = \sum_{\substack{i \in U(k) \wedge \\ u_i^k = 1}} \left(\sum_p X_{ip}(m_{ip}^k + (1 - m_{ip}^k)N_p) + \sum_j Y_{ij} \right) CM$$

In selecting the best alternative for transaction excution the term $CC(T^k, S')$ should be added to $AC(T^k, S')$ and $TC(T^k, S')$.

5.8 Introduction of Storage Limitations in the Design

Storage limitations at each database site may also be introduced into the model. In modern systems storage costs may become a minor component of overall system cost. However, with replication of data it may be useful to include in the optimization constraints which account for storage limitations in order not to exceed available storage at each site. Such a constraint is

$$\sum_i \left(\sum_{p,q | alloc(i,p,q) = j} X_{ip}\, fr(i,p,q) + Y_{ij} \right) size(i) \leq SC_j \quad \text{for every site } j$$

where SC_j measures the storage capacity at site j.

6 EXAMPLE

An example of an application of the optimal non-redundant *allocation model* is shown in Figs. 10 and 11. The example is quite simple, but it incorporates most of the features which are typical of a database distribution problem.

6.1 Description of Requirements

The requirements for the optimization model are described in Fig. 10a, b, and c. We are considering a fully-connected, distributed database consisting of three sites. The *database schema* which has to be distributed (Fig. 10a) consists of three objects (DEPARTMENT, EMPLOYEE, and PROJECT) and three links (DEPT-PROJ, DEPT-EMPL, and PROJ-EMPL). Quantitative parameters and the index numbers associated with objects and links are also shown in the figure. Note that the link between DEPARTMENTs and PROJECTs is included in the logical schema, but is never used by the transactions which are considered; hence it is not subjected to distribution optimization.

The *candidate partitionings* are shown in Fig. 10b. Partitionings 1 and 2 are primary on DEPARTMENT and derived on EMPLOYEE via link DEPT-EMP; partitioning 3 is primary on PROJECT amd derived pm EMPLOYEE via link PROJ-EMP; partitioning 4 is primary on EMPLOYEE. Because of these definitions, the object EMPLOYEE can be partitioned according to 4 alternative candidates (3 derived partitionings and 1 primary partitioning); consequently, variables x_{11}, x_{12}, x_{13}, and x_{14} are introduced. The object DEPARTMENT can be partitioned according to 2 alternative candidates (both primary); correspondingly, variables x_{21} and x_{22} are introduced. Finally, the object PROJECT can be partitioned accordingly to only 1 candidate partition, which is primary (variable x_{33}).

The *Transactions* are shown in Fig. 10c; quantitative parameters are described through Transaction Specification Tables, as in Fig. 7. Transaction 1 is issued at Site $S1$ and matches predicate p_4. Therefore, if the employee OBJECT accessed by the transaction were partitioned according to p_4 in the optimal solution determined by the optimization model, all accesses of $T1$ will be local to Site 1. (This, of course, will also occur if Object 1 is allocated at Site 1 as a whole). Transaction 2 can be issued from all the sites, but it matches predicate $p1$ when it is issued from Site 1. Transaction 4 can be issued from Sites 1 and 2 and in both cases matches partition $P2$.

6.2 The Optimization Model

The optimization model for the example involves the use of 25 variables; in fact, there are 7 X_{ip} variables (corresponding to 4 primary and 3 derived partitionings), 3 W_{hp} variables (for the links along which derived partitions are propagated), 9 variables Y_{ij} (each object can be allocated on each node on a whole), 6 variables V_{hj} (2 links are used by transactions, and can be local to each site). Moreover, 21 constraints are introduced (3 non-redundancy constraints, and 18 constraints for modelling dependencies between variables W_{hp} and X_{ip} or V_{hj} and Y_{ij}. The optimization was made using the Additive Generalized Balas Algorithm, which is a general, 0-1 integer linear solution method; the program was taken from [BaCC81]. The CPU time used for an optimization run was between 1 and 3 seconds on a DEC-20/60 system.

6.3 Discussion of Results

Figure 11 shows seven results of the optimization, obtained by varying the cost parameters and the transaction frequencies. For simplicity, in the case of multiple site transaction, the same frequency value is assigned to each version. A solution is represented in the table by the variables which are set to 1; object or link variables are shown in separate columns.

Cases 1, 2, and 3 show the effect of increasing transmission cost. With null transmission cost ($TC = 0$), each object is allocated *by itself*, and no V_{kj} or W_{kp} variable appears in the solution (as join transmission costs are not evaluated). However, by increasing transmission cost, the object allocation moves to site 1. This is because, in the example, most transactions are issued from site 1, and therefore this solution maximizes the locality of processing.

Cases 4 to 7 show the effect of increasing one of the transaction frequency values in turn. Cases 4 and 5 still maintain the allocation of objects on site 1, because transactions 1 and 2 are issued from 1. However, case 6 presents the partitionings of objects 1 and 3, which are used by transaction 3, according to partitioning 3, which matches the transaction. Likewise, case 7 presents the partitioning of objects 1 and 2 according to partitioning 2, which is matched by transaction 4.

In case 8, the access costs are made equal to 1, without distinguishing local versus remote and retrieval versus update accesses. Then, the objects *move* to site 3, where most of retrievals take place.

Finally, cases 9 and 10 show the effect of increasing the access cost at site 1. In case 9, the objects *move* to site 2; in case 10, the frequency of transaction 3, issued from site 3, is increased, and consequently the objects *move* to site 3. While the behavior of the allocation optimizer can be easily understood and connected a *posteriori*, the allocation chosen by the model is not at all obvious a *priori*.

7 CONCLUSIONS AND FUTURE WORK

In this report we have dealt with the problem of distributing a database by considering the logical schema of the database consisting of objects and predefined links, where individual objects were in the form of normalized relations. By defining four basic logical access types and using a rather straightforward model of transaction execution, we were able to develop necessary equations to estimate costs of transaction processing and develop an optimization model to minimize the costs. A decomposition model was developed to make the problem computationally feasible and a heuristic procedure was discussed to incorporate redundant allocation of entire objects or partitioned objects.

This paper has developed a methodology for the distribution design phase (see Fig. 1) which fits in the overall framework of database design. This phase is into a series of activities which are necessary before the optimization model can in fact be applied. An important early step is the solicitation of partioning guidance from the user, which makes this model tractable. The example in Section 6 points out the scenario of a distribution design by considering various possible mixes of transactions.

It is conceivable that after a database has been distributed and is operational a restructuring is called for, due to the following reasons:

i Better transaction load estimates are available.

ii There is a need to introduce new objects and links in the database schema and repopulate the database.

iii Cost parameters such as access costs and transmission costs have undergone a change.

The approach that can be taken to deal with the above problem is to run a revised optimization model. The revision consists of adding to each C_{ip} or D_{ij} the cost of moving the already allocated object to the intended new location. This cost should be averaged over the time period between two restructuring since all other costs are per unit time. The non-redundant model could then be run with these new cost parameters.

Redoing the problem under redundancy amounts to solving the old and the new problems with new parameters and advocating restructuring if

$$C_{old} - C_{new} > \frac{R}{P}$$

where, C_{old} and C_{new} are total costs of transaction processing for the old and new allocations using new parameters.

R is the cost of a one-time restructuring

P is the time units between two restructurings

A sophisticated database environment with a built-in design tool which is capable of doing the above type of restructuring can be expected to monitor loads and trigger a distribution of data over the network when needed.

Future extensions to this work will address:

i A vertical partitioning of objects.

ii Different models for transaction execution.

iii More general models for description of data replication; *e.g.* the effectiveness of redundantly allocating a single fragment.

iv Consideriation of equivalent logical schemas for further optimization; *e.g.* when a subproblem is decomposed, alternative logical schema representations of the subproblem should be investigated.

The current model is encouraging since it permits a formal solution to a design problem which is too complex to be solved by random search and for which no good directed search algorithms are known. At the same time few people have the experience to design distributed databases by intuition, although the results obtained from testing our model were explainable in informal terms.

Acknowledgements

This work was performed at Stanford University as part of the Knowledge Base Management Systems project, supported by Defense Advanced Research Projects Agency contract N39-80-G-0132. Stefano Ceri was partially supported by a grant from the Italian National Research Council and by grant AFOSR-80-0212 in accordance with NSF agreement IST-80-21358 to prof. J.D. Ullman at Stanford University. Many collegues have participated in discussions on distributed systems and inspired some of the concepts used here. We wish to thank them all.

REFERENCES

[BaCC81] C. Baldissera, S. Ceri and A. Colorni: "Optimization Methods and Computer Programs"; *Informatica Series*, CLUP.Ed, 1981 (in Italian).

[BeSR80] P.A. Bernstein, D.W. Shipman and J.B. Rothnie: "Concurrency Control in a System for Distributed Databases (SDD-1)"; *ACM TODS*, vol. 5, no. 1, March 1980.

[Case72] R.G. Casey "Allocation of Copies of a File in an Information Network"; *Proc. of the 1972 SJCC*, AFIPS vol.40, 1972.

[CeMP80] S. Ceri, G. Martella and G. Pelagatti: "Optimal File Allocation for a Distributed Database on a Network of Minicomputers"; *Proc. International Conference on Databases*, Aberdeen, July 1980, British Computer Society, Hayden Publishers.

[CePe79] S. Ceri and G. Pelagatti: *The Allocation of Operations in Distributed Database Access*; Politechnico Milano Rep. IEEEPM no. 79-20, to appear in IEEE Transactions on Computers.

[CePB81] S. Ceri, G. Pelagatti and G. Bracchi: "Structured Methodology for Designing Static and Dynamic Aspects of Database Applications"; *Information Systems*, vol. 6, no. 1, Jan. 1981.

[ChAk80] P.P.S. Chen and J. Akoka: "Optimal Design of Distributed Information Systems"; *IEEE-TC*, vol. C-29, no. 12, Dec. 1980.

[ChCh80] S.K. Chang and W.H. Cheng: "A Methodology for Structured Database Decomposition"; *IEEE-TSE*, vol. SE-6, no. 2, Mar. 1980.

[Chen76] P.P.S. Chen: "The Entity-Relationship Model - Towards a Unified View of Data"; *ACM TODS*, vol. 1, no. 1 Mar. 1976.

[ChHe76] K.M. Chandy and J.E. Hewes: "File Allocation in Distributed Systems "; *Proc. Int. Symp. on Computer Performance*, Modeling, Measurement, and Evaluation, Mar. 1976, pp. 10-13.

[Chu69] W.W. Chu: "Optimal File Allocation in a Multiple Computer System"; *IEEE Transactions on Computers*, vol. C-18, no. 10, 1969.

[Codd74] E.F. Codd: "Recent Investigations into Relational Database Systems"; *Proc. IFIP Congress*, North Holland, 1974.

[ElWi79] R. El-Masri and G. Wiederhold: "Data Model Integration Using the Structural Model"; *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Boston, 1979.

[Eswa74] K. P. Eswaran: "Placement of Records in a File and File Allocation in a Computer Network"; *Proc. IFIP Congress*, North Holland, 1974.

[HaNi79] M. Hannes and B. Niami: "A Heuristic Approach to Attribute Partitioning"; *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Boston, 1979.

[HeYa79] A.R. Hevner and S.B. Yao "Query Processing in Distributed Databases "; *IEEE Trans. on Software Engineering*, vol. SE-5, no. 3, 1979.

[Hoff76] J.A. Hoffer: "An Integer Programming Formulation of Computer Database Design Problems"; *Information Science*, Vol. 11 (July 1976), pp. 29-48.

[HoSe75] J.A. Hoffer and D.G. Severance: "The Use of Cluster Analysis in Physical Database Design "; *Proc. First Int. Conf. on Very Large Databases*, Framingham MA, 1975.

[IrKh79] K.B. Irani and N.G. Khabbaz "A Mod̲el for a Combined Communication Network Design and File Allocation for Distributed Databases"; *Proc. First Int. Conf. on Distributed Computing Systems*, 1979.

[LumA79] V.Y. Lum *et al* "1978 New Orleans Database Design Workshop Report"; *Proc. Fifth Int. Very Large Database Conference*, Rio de Janeiro, Brazil (Oct. 1979).

[MoLe77] H.L. Morgan and J.D. Levin: "Optimal Program and Data Locations in Computer Networks"; *CACM*, vol. 20, no. 5, 1977.

[MaRi76] S. Mahmoud and J.S. Riordon: "Optimal Allocation of Resources in Distributed Information Networks"; *ACM TODS*, vol. 1, no. 1, 1976.

[MiWi81] T. Minoura and G. Wiederhold: "Resilient Extended True-Copy Token Scheme for a Distributed Database System"; *Proceedings of the IEEE Symposium on Reliability in Distributed Software and Database Systems*, July 1981

[Nava80] S.B. Navathe: "Schema Analysis for Database Restructuring"; *ACM TODS*, vol. 5, no. 2, (June 1980), pp. 157-184.

[NaCW81] S.B. Navathe, S. Ceri and G. Wiederhold*A Methodology for Distributed Database Design* ; working paper (in preparation), University of Florida, Gainesville.

[NaSc78] S.B. Navathe and M. Schkolnick: "View Representation in Logical Database Design"; *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, 1978, pp. 144-156.

[RaWa79] C.V. Ramamoorthy and B.W. Wah: "The Placement of Relations on a Distributed Relational Database"; *Proc. First Int. Conf. on Distributed Computing Systems*, 1979.

[SuLL81] Y. W. Su, H. Lam and D. H. Lo: "Transformation on Data Traversals and Operations in Application Programs to Account for Semantic Changes of Databases"; *ACM TODS*, vol. 6, No. 2, June 1981.

[TeFr80] T.J. Teorey and J.P. Fry: "The Logical Record Access Approach to Database Design"; *ACM Computing Surveys*, vol. 12, no. 2, June 1980.

[Whit70] V.K.M. Whitney: *A Study of Optimal File Assignment and Communication Network Configuration in Remote-Access Computer Message Processing and Communications Systems* ; Ph.D. Dissertation, University of Michigan, 1970.

[WiEl80] G. Wiederhold and R. El-Masri: "The Structural Model for Database Design"; *The Entity Relationship Approach to System Analysis and Design*, (Chen, ed.), North-Holland 1980, pp. 237-257.

[Wong77] E. Wong: "Retrieving Dispersed Data from SDD-1, A System for Distributed Databases"; *Proc. Berkeley Workshop on Distributed Data Management and Computer Networks*, 1977.

[YaNW78] S.B. Yao, S.B. Navathe and J.L. Weldon: "An Integrated Approach to Logical Database Design"; *Proc. NYU Symposium on Database Design*, 1978, NYU GSB.

EXAMPLES AND FIGURES

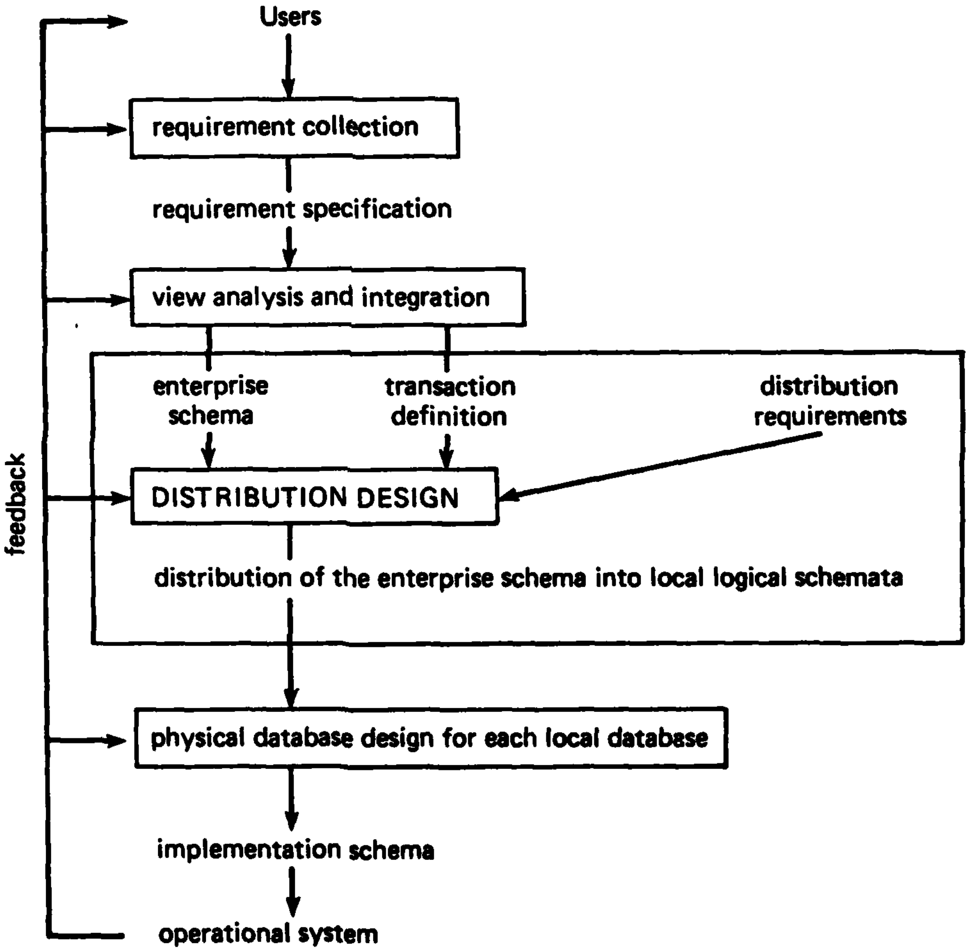
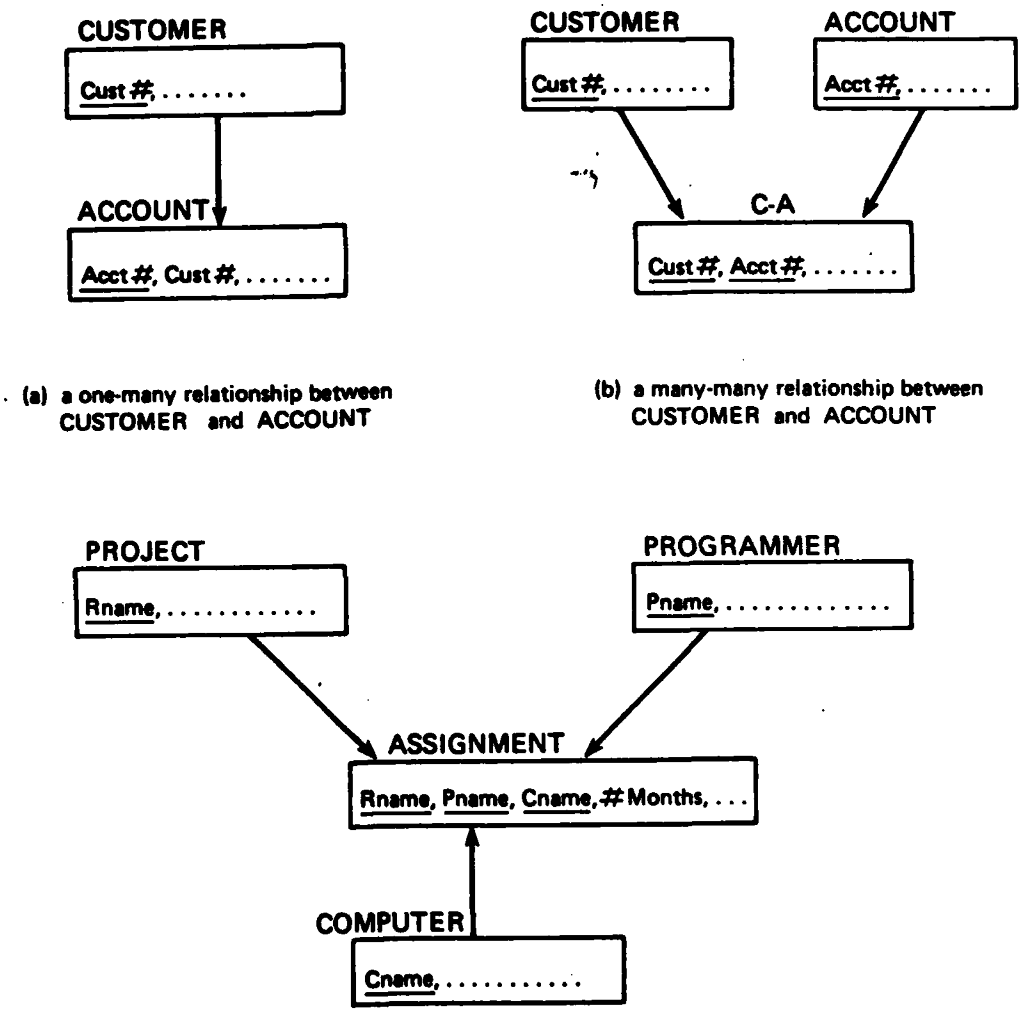

Users
requirement collection
requirement specification
view analysis and integration
enterprise schema
transaction definition
distribution requirements
DISTRIBUTION DESIGN
distribution of the enterprise schema into local logical schemata
physical database design for each local database
implementation schema
operational system
feedback

Figure 1 The Overall Distributed Database Design Methodology


CUSTOMER
Cust #, ...
ACCOUNT
Acct #, Cust #, ...
CUSTOMER
Cust #, ...
ACCOUNT
Acct #, ...
C-A
Cust #, Acct #, ...
PROJECT
Rname, ...
PROGRAMMER
Pname, ...
ASSIGNMENT
Rname, Pname, Cname, # Months, . . .
COMPUTER
Cname, ...

(a) a one-many relationship between CUSTOMER and ACCOUNT

(b) a many-many relationship between CUSTOMER and ACCOUNT

(c) a ternary relationship between PROJECT, PROGRAMMER and COMPUTER

Figure 2 Use of Objects and Links to Model Different Types of Relationships

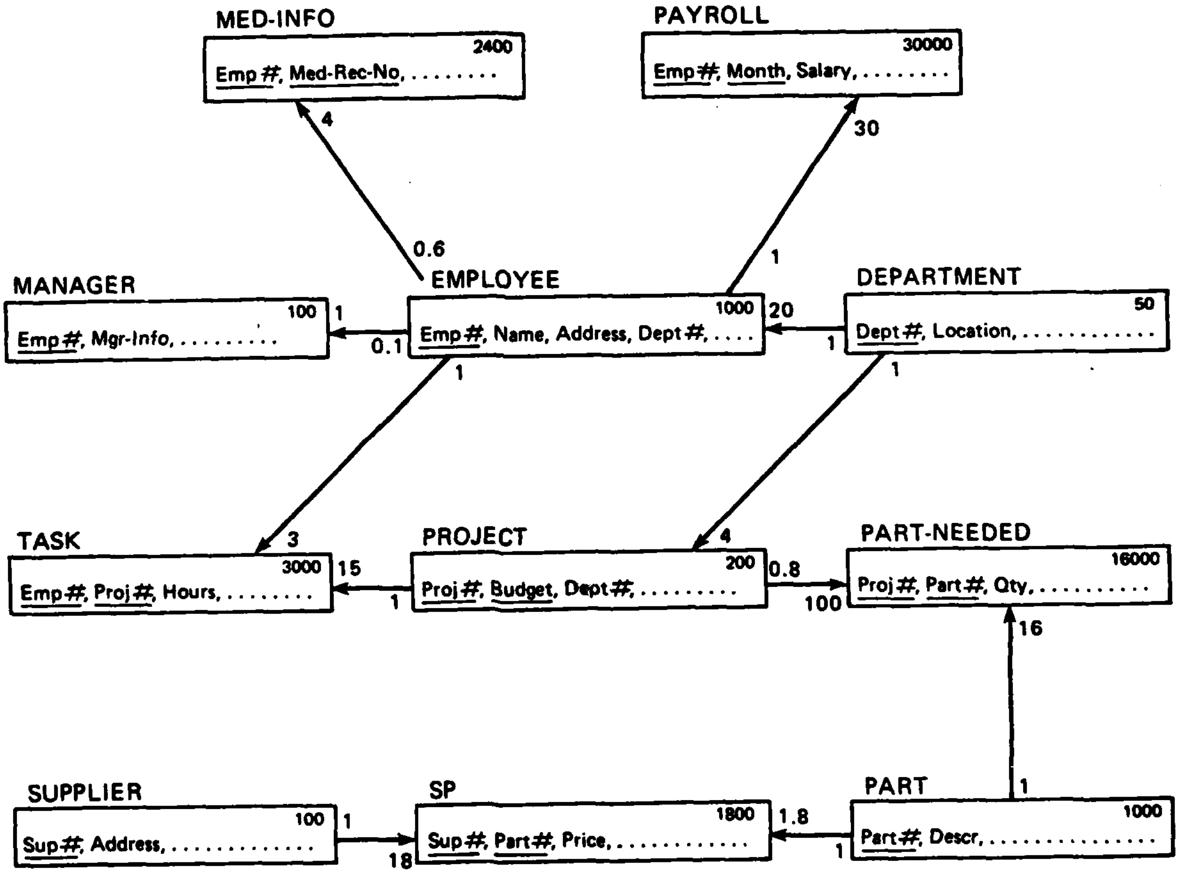
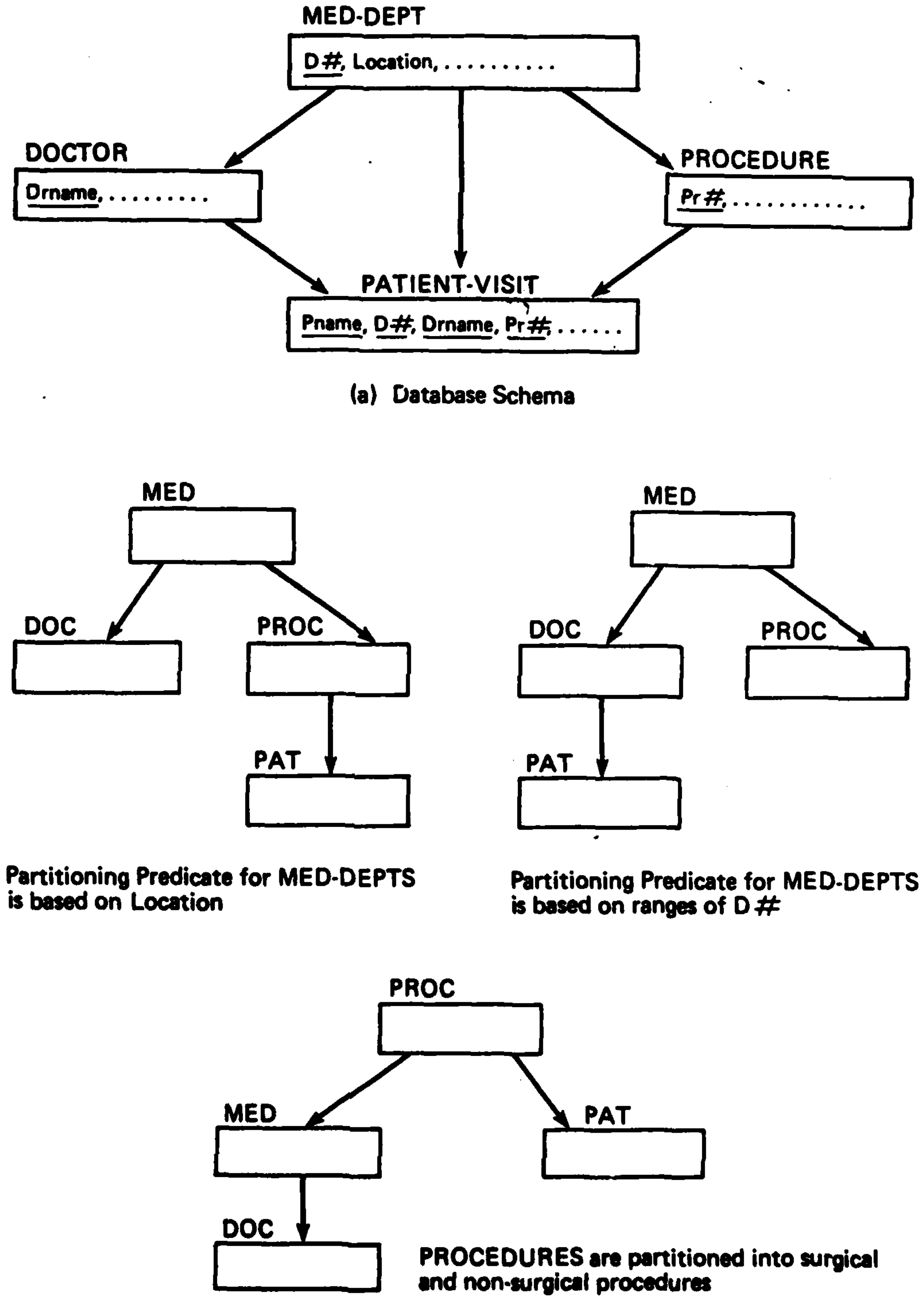

MED-INFO
2400
Emp #, Med-Rec-No, ...
PAYROLL
30000
Emp #, Month, Salary, ...
4
30
0.6
1
MANAGER
100
Emp #, Mgr-Info, ...
1
0.1
EMPLOYEE
1000
Emp #, Name, Address, Dept #, ...
20
1
DEPARTMENT
50
Dept #, Location, ...
1
1
TASK
3000
Emp #, Proj #, Hours, ...
3
15
1
PROJECT
200
Proj #, Budget, Dept #, ...
4
0.8
100
PART-NEEDED
16000
Proj #, Part #, Qty, ...
16
1
SUPPLIER
100
Sup #, Address, ...
1
18
SP
1800
Sup #, Part #, Price, ...
1.8
1
PART
1000
Part #, Descr, ...

Figure 3 An Example of a Database Schema


MED-DEPT
D#, Location, ...
DOCTOR
Drname, ...
PROCEDURE
Pr#, ...
PATIENT-VISIT
Pname, D#, Drname, Pr#, ...
MED
DOC
PROC
PAT
MED
DOC
PROC
PAT
PROC
MED
PAT
DOC

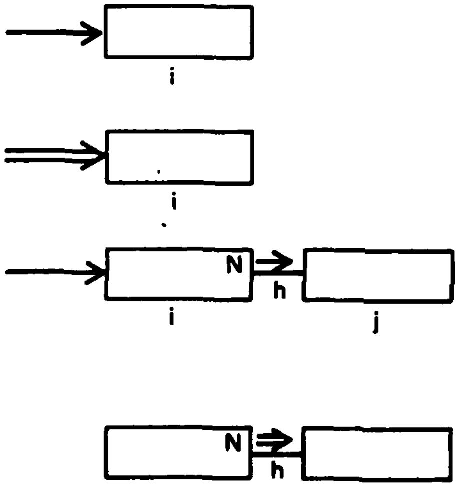
(a) Database Schema

Partitioning Predicate for MED-DEPTS is based on Location

Partitioning Predicate for MED-DEPTS is based on ranges of D#

PROCEDURES are partitioned into surgical and non-surgical procedures

Figure 4 A Database Schema and Its Possible Derived Partitioning Hierarchies



TDA: Total Direct Access.
no. accessed = card (i)

SDA: Selective Direct Access.
no. accessed is user specified

TJA: Total Join Access.
no. accessed = *N * ratio, if j is*
the member of h
= N, if j is the owner
of h

SJA: Selective Join Access.
no. accessed is user specified

Figure 5 Transaction Access Primitives

Transaction: Find all EMPmployees in the DEPartments in CAlifornia that have PROJects which need PART#= 7386.
List EMP#, DEPT#, PROJ#, BUDGET.

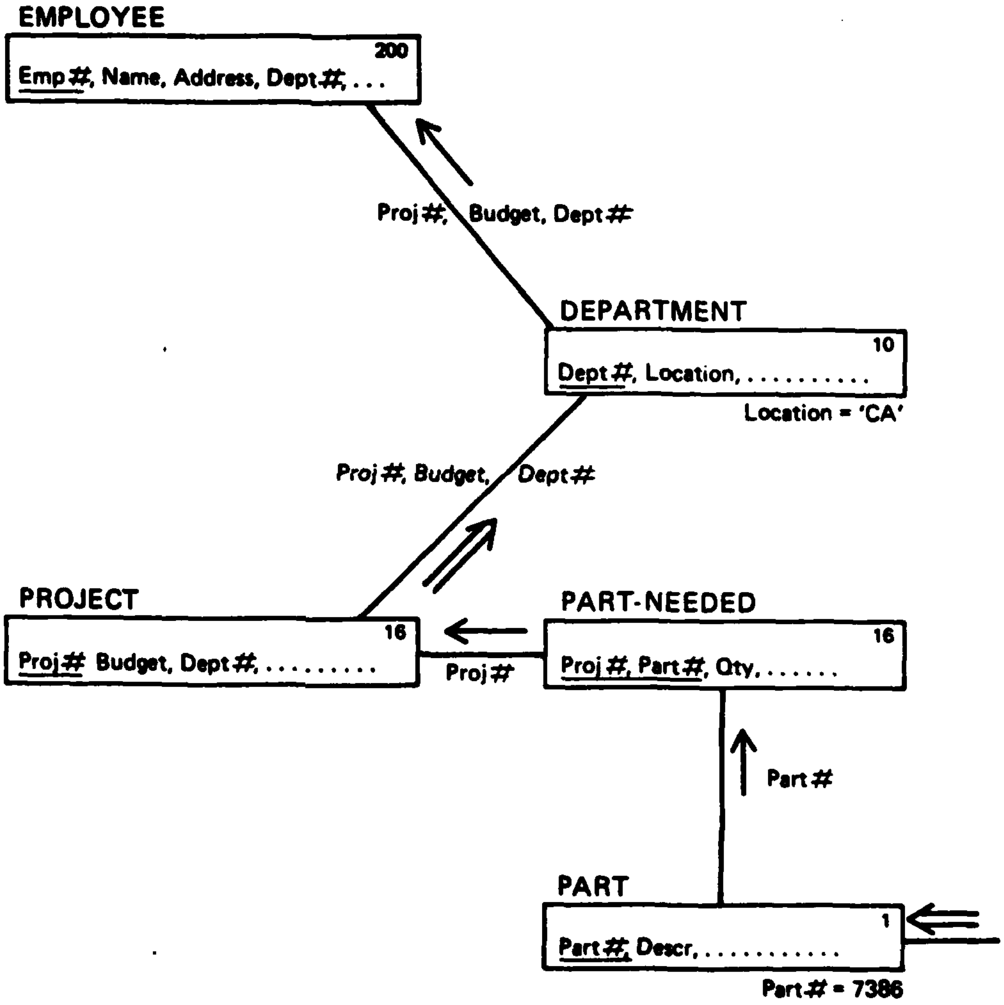

EMPLOYEE
200
Emp #, Name, Address, Dept #, . . .
Proj #, Budget, Dept #
DEPARTMENT
10
Dept #, Location, ...
Location = 'CA'
Proj #, Budget, Dept #
PROJECT
16
Proj # Budget, Dept #, ...
Proj #
PART-NEEDED
16
Proj #, Part #, Qty, ...
Part #
PART
1
Part #, Descr, ...
Part # = 7386

Figure 6 A Graphical Transaction Specification

TRANSACTION SPECIFICATION

Object	Used as	Used	No.of	Link	Partn'g	Size of	Retrvl	Evaluated
	Entry	for	Acces'd	Used	Pred.	Tuple	vs	no. Bytes
	Point	Result	Tuples	Next	Matched	X'mitted	Update	X'mitted
part	1	0	1	P-PN	-	4	R	4
part-needs	0	0	16	PN-PR	-	4	R	64
project	0	0	16	PR-D	P1	12	R	192
department	0	0	10	D-E	P1	12	R	10·12=120
employee	0	1	200	-	P1	16	R	200

Origin at: Site 3 Freq$_{33}$: 50/month

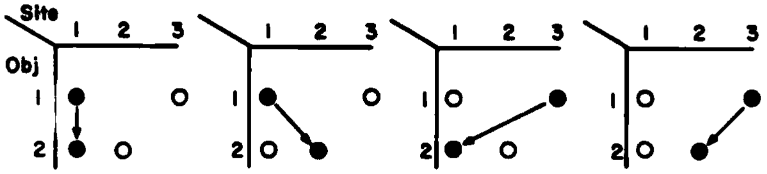
Figure 7 A Tabular Transaction Specification

Let $S = y_{11}, y_{13}, y_{21}, y_{22}$ represent a redundant allocation of a database having 3 sites and 2 objects, where all non-mentioned variables are set to zero. Cases a, b, c, d of Section 5.2 are represented below with diagrams, where solid dots correspond to objects in $SOL(S, I^k)$; arrows indicate precedences between transaction accesses; and a vertical arrow connects accesses performed at the same site, which do not require any transmission.

Case a: Retrieval-Retrieval

$$I^k = \{O_1, O_2\}, SOL(S, I^k) = \{\{y_{11}, y_{21}\}, \{y_{11}, y_{22}\}, \{y_{13}, y_{21}\}, \{y_{13}, y_{22}\}\}$$

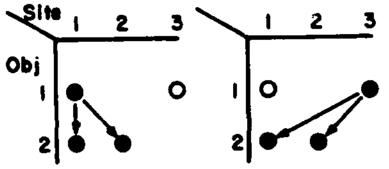
A different transaction execution is obtained from each solution in $SOL(S, I^k)$;



Case b: Retrieval-Update

$$I^k = \{O_1\} = SOL(S, I^k) = \{\{y_{11}, y_{21}, y_{22}\}, \{y_{13}, y_{21}, y_{22}\}\}$$

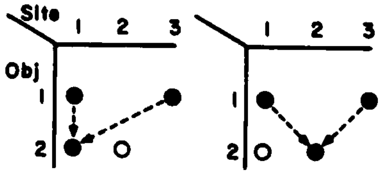
Transmissions to both copies of the updated object are required.



Case c: Update-Retrieval

$$I^k = \{O_2\} = SOL(S, I^k) = \{\{y_{11}, y_{13}, y_{21}\}, \{y_{11}, y_{13}, y_{22}\}\}$$

Transmission is required from one of the copies of the updated object to the retrieved object; the best transmission alternative is selected.



Case d: Update-Update

$$I^k = \phi, SOL(S, I^k) = \{\{y_{11}, y_{13}, y_{21}, y_{22}\}\}$$

Transmission to both copies of Object 2 are required for both copies; the best alternative is selected.

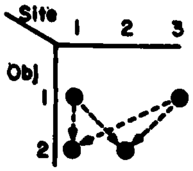


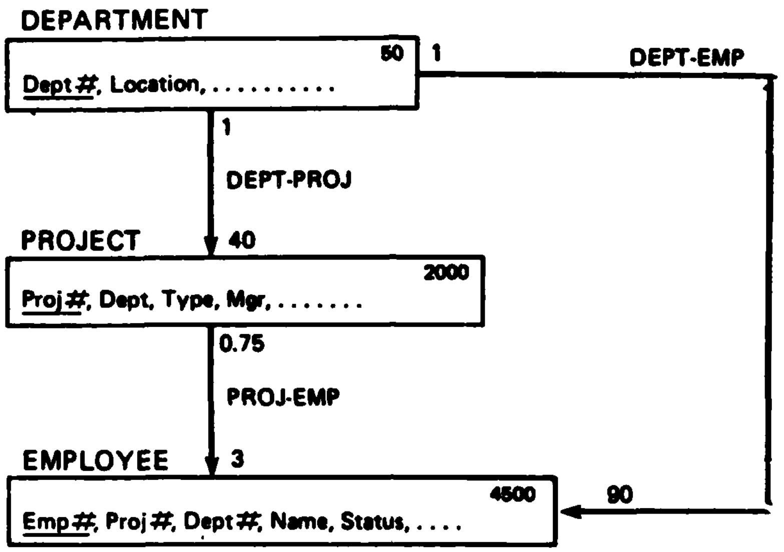
Figure 8 Transaction Execution over a Redundantly Distributed Database

```
Start with $S_{new} = S^*$, $C_{new} = C^*$
                        (* refers to the optimal non-redundant solution)
  repeat
     $S := S_{new}$; $C := C_{new}$;
     for every variable $X_{ip}$ or $Y_{ip}$ which does not belong to $S$ do
     begin
        build $S' = (S$ +that variable set to 1);
        $C' := 0$;
        for every transaction $T^k$ such that $i \in U(T^k)$ do
        begin
          compute $CT(T^k, S')$;
          $C' := C' + CT(T^k, S')$;
        end;
        for every transaction $T^k$ such that $i \not\in U(T^k)$ do
          $C' := C' + CT(T^k, S)$;
        $C' := C' + \sum_i f(dr_i)$;
        if $C' < C_{new}$ then
             begin $S_{new} := S'$; $C_{new} := C'$ end
     end
  until $(C_{new} < C)$;
```

S, S', S_{new} represent assignments of variables X_{ij} and Y_{ij} corresponding to old, partial and new solution at each iteration. C, C', C_{new} are the corresponding total costs. The counts dr_i are the degree of redundancy of object i in solution S.

Figure 9 Algorithm for Redundant Database Distribution

(a) Database Schema


DEPARTMENT
50
1
DEPT-EMP
Dept#, Location, ...
1
DEPT-PROJ
PROJECT
40
2000
Proj#, Dept, Type, Mgr, ...
0.75
PROJ-EMP
EMPLOYEE
3
4500
90
Emp#, Proj#, Dept#, Name, Status, ...

Objectname	Index
Employee	1
Department	2
Project	3

(b) Candidate Partitionings

P1: *primary* on DEPARTMENT (Object 2)

pred(2, 1, 1) : DNO in {1...20}; *alloc*(2, 1, 1) = 1; *fr*(2, 1, 1) = 0.4
pred(2, 1, 2) : DNO in {21...50}; *alloc*(2, 1, 2) = 3; *fr*(2, 1, 2) = 0.6
derived on EMPLOYEE via link DEPT-EMP

P2: *primary* on DEPARTMENT

pred(2, 2, 1) : LOCATION=Northern California; *alloc*(2, 2, 1) = 1; *fr*(2, 2, 1) = 0.26
pred(2, 2, 2) : LOCATION=Central California ; *alloc*(2, 2, 2) = 2; *fr*(2, 2, 2) = 0.30
pred(2, 2, 3) : LOCATION=Southern California; *alloc*(2, 2, 3) = 3; *fr*(2, 2, 3) = 0.44
derived on EMPLOYEE via link DEPT-EMP

P3: *primary* on PROJECT

pred(3, 3, 1) : TYPE=Software; *alloc*(3, 3, 1) = 1; *fr*(3, 3, 1) = 0.7
pred(3, 3, 2) : TYPE=Hardware; *alloc*(3, 3, 2) = 3; *fr*(3, 3, 2) = 0.3
derived on EMPLOYEE via link PROJ-EMP

P4: *primary* on EMPLOYEE

pred(1, 4, 1) : STATUS=Regular ; *alloc*(1, 4, 1) = 2; *fr*(1, 4, 1) = 0.5
pred(1, 4, 2) : STATUS=Part-time; *alloc*(1, 4, 2) = 2; *fr*(1, 4, 2) = 0.2
pred(1, 4, 3) : STATUS=Fired ; *alloc*(1, 4, 3) = 3; *fr*(1, 4, 3) = 0.3

Figure 10 Example of a Database Schema, Candidate Partitionings, and Transactions

(c) Transactions

T1: Give 5% raise to salaries of regular employees

ORIGIN: Site 1 (FREQUENCY: $freq_{11}$)

Object	Used as Entry Point	Used for Result	No.of Acces'd Tuples	Link Used Next	Partn'g Pred. Matched	Size of Tuple X'mitted	Retrvl vs Update	Evaluated no. Bytes X'mitted
EMPLOYEE	Yes	Yes	2250	--	P4	-	U	--

T2: List name and salary of employees with department city = San Francisco

ORIGIN: Site 1 (FREQUENCY: $freq_{22}$)

DEPARTMENT	Yes		5	DEPT-	P1	4	R	20
EMPLOYEE		Yes	450	EMP	P1	40	R	1800

ORIGIN: Site 2 (FREQUENCY: $freq_{22}$) and Site 3 (FREQUENCY: $freq_{23}$)

DEPARTMENT	Yes		5	DEPT-		4	R	20
EMPLOYEE		Yes	450	EMP		40	R	1800

T3: List all participants to hardware projects whose manager is Jones

ORIGIN: Site 3 (FREQUENCY: $freq_{33}$)

PROJECT	Yes		20	PROJ-	P3	4	R	80
EMPLOYEE		Yes	45	EMP	P3	40	R	180

T41: Give a 5% raise to employee salaries for the departments in Northern California

ORIGIN: Site 1 (FREQUENCY: $freq_{41}$)

DEPARTMENT	Yes		13	DEPT-	P2	4	R	52
EMPLOYEE		Yes	1170	EMP	P2	-	U	--

T42: Give a 5% raise to employee salaries for the departments in Central California

ORIGIN: Site 2 (FREQUENCY: $freq_{42}$)

DEPARTMENT	Yes		15	DEPT-	P2	4	R	60
EMPLOYEE		Yes	1350	EMP	P2	-	U	--

Figure 10 Example of a Database Schema, Partitionings, and Transactions (continued)

TABLE OF RESULTS

Case	Cost Parameters		Transaction Frequency	Optimal Solutions		Cost
	CLR,CRR,CLU,CRU	TC	$f_{1...4}$	y and	w	
1	$CLR=1,CRR=2.5,$ $CLU=5,CRU=10$	0	$f_1 =$ $f_2 = f_3 =$ $f_4 = 1$	y_{11}, x_{22}, y_{33}	—	33490.50
2		0.5		y_{11}, y_{21}, y_{33}	$w_{11},$ w_{12}	33553.00
3		1				33563.00
4			$f_1 = 100, f_{2,3,4} = 1$			1147493.00
5			$f_2 = 100, f_{1,3,4} = 1$	y_{11}, y_{21}, y_{31}		305813.25
6			$f_3 = 100, f_{1,2,4} = 1$	x_{13}, y_{23}, x_{33}	v_{23}	40916.00
7			$f_4 = 100, f_{1,2,3} = 1$	x_{12}, x_{22}, y_{33}	v_{12}	1289025.00
8	$CLR=CRR=$ $CLU=CRU=1$		$f_1 = f_2 =$ $f_3 = f_4 = 1$	y_{13}, y_{23}, y_{33}	$w_{13},$ w_{23}	6243.00
9	$CLR_1=10,CRR_1=25,$ $CLU_1=50,CRU_1=100$		$f_1 = f_2 =$ $f_3 = f_4 = 1$	y_{12}, y_{22}, y_{23}	$w_{12},$ w_{22}	43910.00
10	$CLR_{2,3}=1,CRR_{2,3}=25,$ $CLU_1=5,CRU_1=100$		$f_3 = 100,$ $f_1 = f_2 = f_4 = 1$	y_{13}, y_{23}, y_{33}	$w_{13},$ w_{23}	50702.00

Figure 11 Optimal Non-Redundant Solution for Several Values of the Cost Parameters and the Transaction Frequencies

DATE
ILMED

8