DISTRIBUTING A DATABASE FOR PARALLELISM

by

E. Wong and R.H. Katz

Memorandum No. UCB/ERL M82/86

6 October 1982

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

DISTRIBUTING A DATABASE FOR PARALLELISM

E. Wong and R.H. Katz*
Department of Electrical Engineering and Computer Sciences
and the Electronics Research Laboratory
University of California, Berkeley, California 94720

ABSTRACT

In this paper we treat the problem of subdividing a database and allocating the fragments to the sites in a distributed database system in order to maximize non-duplicative parallelism. Our goal is to establish a conceptual framework for distributing data without being committed to specific cost models.

We introduce the concept of "local sufficiency" as a measure of parallelism, and show how certain classes of queries lead naturally to irredundant partitions of a database that are locally sufficient. For classes of queries for which no irredundant distribution is locally sufficient, we offer ways to introduce redundancy in achieving local sufficiency.

---

*University of Wisconsin, Madison, Wisconsin.

1. Introduction

The context in which the problem arises is that of a distributed database system. By this we mean any system consisting of multiple autonomous processors communicating through a communication medium and each accessing a separate fragment of the database, and where the collection of fragments is to be seen by the user as an integrated whole. Geographical dispersion is not a necessary ingredient. Replication and, more generally, redundancy of data among the fragments may be present. The question we pose is the following: if we are free to subdivide the database into possibly overlapping fragments, how should we do it?

The objective of the subdivision is efficient performance for both retrieval and update operations. While it would not be difficult to express the problem of subdividing a database as one of minimizing a weighted cost of database usage, doing so is not particularly useful for several reasons:

(1) Weighting of costs requires knowing the frequenices of usage for different operations. Such statistics are neither easy to obtain nor very stable. They are better used in fine tuning than in the basic structural design.

(2) Similarly, for a specific database operation, an appropriate cost function is not easily estimated. In particular, how the cost depends on the partition-p9licy is not likely to be known quantitatively.

(3) Even if precise cost could be computed for each partition, there would be too many ways of subdividing a database for the problem of finding the minimum to be tractable.

These considerations suggest that what is needed is a conceptual framework that captures and makes precise qualitative factors important

in designing a database partition. The goal is not to find a single strategy, but to identify classes of strategies with desirable properties. Within each class fine-tuning can then take place. The challenge is to do as much as one can in the design without having to use quantitative design data. In this our approach differs significantly from the existing work on file allocation [CHU 73, MAHO 76] where a quantitative cost model plays a central role.

2. Local Sufficiency and Minimal Redundancy

Let $\mathcal{D}$ denote the database as seen by the user. Neither distribution nor redundancy is visible in $\mathcal{D}$. Let M_i denote the fragment of the database associated with the i^{th} processor. We assume that $\bigcup_i M_i = \mathcal{D}$ and call the collection $M = \{M_i\}$ a materialization of $\mathcal{D}$. The problem of partitioning is to find the "best" materialization.

Let Q denote a class of queries on $\mathcal{D}$. We shall say a materialization M is locally sufficient (relative to Q) [WONG 81] if for every q Q there exist local queries q_i on M_i such that

$$\text{result}(q,\mathcal{D}) = \bigcup_i \text{result}(q_i, M_i) \qquad (2.1)$$

Local sufficiency means that no communication among the processors is needed to process a query in Q, the only data movement being a final one to collect the fragments of the result produced at different sites.

Local sufficiency is clearly a desirable property for retrieval operations. It is in general not attainable without redundancy, and that imposes a cost on updates. The tension thus created makes the design problem interesting.

For any two materializations M and M' (of the same $\mathcal{D}$) define a partial ordering $M > M'$ by

$$M > M' \iff M_i \supseteq M'_i \qquad \text{for every } i \tag{2.2}$$

If $M > M'$, then (2.2) implies

$$M_i \cap M_{ij} \supseteq M'_i \cap M'_j \qquad \text{for all } i,j$$

Hence, $M > M'$ means that M is at least as redundant as M'.

Let M be a locally sufficient materialization for a given Q. We say that M is <u>minimally redundant</u> (Q) if for every $M' < M$, M' being Q-locally sufficient implies $M' = M$. In other words, M being minimally redundant means there is no locally sufficient materialization that is less redundant than M.

It is reasonable to assume that a query takes longer to process if the volume of data is greater. For example, in a relational system the processing time for a restriction, projection or join is a non-decreasing function of the cardinalities of the relations involved, regardless of what storage structures and processing algorithms are used. Under such an assumption, a minimally redundant M is always better than one that is not, for any $q \in Q$.

It is perhaps even more reasonable to assume that the cost of an update is a nondecreasing function of the degree of redundancy, whatever the underlying implementation. This is so because updating with redundancy is tantamount to an update without redundancy plus the enforcement of an integrity constraint. Such enforcement never comes free. It follows that in designing a materialization, we can limit our choices to those that are minimally redundant, whatever the underlying implementation and physical conditions. In so doing, we have succeeded in extracting from a rather complex design problem an approach for optimization that is nearly universally applicable becasue it is free from quantitative assumption.

It is interesting to note that any materialization M that contains full replicas of the database at two or more sites can never be minimally redundant, because an irredundant M', having the entire database at a single site, is already locally-sufficient for any Q, and clearly $M' < M$.

We shall take minimal-redundancy as a criterion of goodness in designing a distributed database. As the example of having all the data at a single site shows, a minimally redundant materialization need not have all the desirable characteristics. One that is not, however, is almost certain to be a bad design. Being entirely qualitative, the criterion of minimum-redundancy is not sufficient to reduce the choice to a single design, but it does achieve a drastic reduction in the number of candidates that need to be considered.

3. A Semantic Model

Minimal redundancy is defined relative to a class of queries Q. How should Q be chosen? Examples quickly suggest that natural query-classes are determined by semantics. Therefore, we shall introduce a simple semantic model, and define retrieval and update operations in terms of this model.

The model that we choose is a simplified version of the entity-relationship model [CHEN 76, WONG 79]. An entity is an undefined atomic object. An entity type is a named collection of entities. A relationship is a "relation" with entity types as its domains. For example, consider a "company" database consisting of the following:

entity types: emp, dept, job

relationship: qualified (emp,job)
assign (emp,dept,job)
mgr (dept,emp)

The participation of an entity type in a relationship may be subject to one or both of the following constraints:

(a) E is unique in R -- Each entity of E can occur at most once in R.

(b) E is total in R -- Each entity of E must occur at least once in R.

If E is both unique and total in R, then R is a function on E . In such a case we shall call R an association. For example, suppose that every employee has a unique assignment of both job and dept, then the relationship assign is an association on emp. On the other hand, suppose each dept has at most one mgr but some dept's may be temporarily without one, then dept is unique in mgr(dept,emp) but not total, and mgr is not an association. We shsll refer to entity types and those relationships that are not associations collectively as primitive objects. An attribute is a function mapping a primitive object into a value set, which is any machine-interpretable data type (e.g., integers, character strings). For example, the following is a description of the "company" database in terms of the semantic objects that we have introduced:

Example 3.1 Semantic Schema for the "Company" Database

entity types: emp,dept,job

non-assoc. relationships: qualified (emp,job)

association: emp $\xrightarrow{\text{assign}}$ dept,job

attributes: emp $\xrightarrow{\text{ename}}$ c20

emp $\xrightarrow{\text{age}}$ i2

dept $\xrightarrow{\text{dname}}$ c10

job $\xrightarrow{\text{title}}$ c10

job $\xrightarrow{\text{pay}}$ i4

qualified $\xrightarrow{\text{year}}$ i2

Suppose that every entity of each type is assigned a unique non-updatable identifier that that serves as a surrogate for the entity. Then, the semantic description of the database can be mapped immediately into a collection of relations free of any serious update anormalies. The basic mapping rule can be stated as: "One and only one relation per primitive object." For example, let eno, dno and jid denote the identifiers of emp, dept and job respectively. Then, the mapping rule yields the following relational schema for the "Company" database:

Exampl3 3.2 Relational Schema of "Company" Database

emp (eno*,ename,age,assign-dno, assign-jid)

dept (dno*, dname)

job (jid*, title, pay)

qualified (eno, jid, year)

mgr (dno, eno)

For a relation representing an entity type, the identifier-domain of that entity type is indicated by an asterisk and will be called its primary key. The underscored domains are identifier domains for entity types represented by some other relations, and they will be referred to as foreign keys. Thus, for example, eno is the primary key of emp and assign-dno is a foreign key in emp.

4. Semantic Queries

Assume that the user's view of the distributed database is given by a relational schema, free of any distribution information and designed according to the procedure given in Section 3. The question we address here is: what are the semantically natural queries?

We say a relational query is a semantic query if:

(a) it is a one-variable query (i.e., it involves a unary operation), or

(b) it is an equijoin of two relations on an identifier domain, or

(c) it involves a finite sequence of operations of types (a) and (b).

The principal restriction that semantic queries must satisfy is that "joins" can only be "equality on identifier domains" (primary or foreign keys). For example, the QUEL query

```
range of e is emp
range of j is job
retrieve (e. ename) where 1000 * e.age > j.salary
```

is not semantic because the join-condition is neither an equality nor on an identifier domains. However, if the condition "e.assign-jid = j.jid" were added to the qualification, the resulting query would be semantic, since it would then involve an equi-join on the "jid" domain, to be followed by a restriction on the condition 1000 * e.age > j.salary.

There are at least two reasons to restrict queries used in designing a distributed database to semantic queries. First, they are more natural, and hence are likely to be representative of the queries used in practice. Second, these queries reflect the semantics of the schema so that the schema can be used to suggest the class of queries to be used in deciding how the database is to be distributed.

5. Semantically Induced Irredundant Materialization

The problem we face at this point is the following: Given a relational schema designed by using the mapping rules of Section 3, how do

T will be called a partition-tree of a schema graph G if:

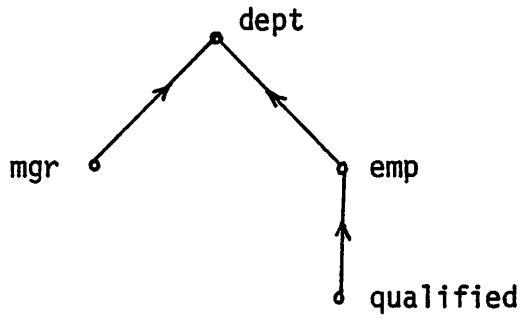
(a) T is a subgraph of G

(b) As an undirected graph, T is a tree.

(c) Each arc in T is directed from son to parent.

For example, the following is a partition-tree of the schema graph of Example 5.1:

Example 5.2



Proposition 5.1. A partition of the root-relation in a partition tree induces a unique partition of every relation in the tree.

Proof: The proposition follows trivially by induction on the depth of the tree and from the property that $R \rightarrow S$ together with a partition of S induce a unique partition of R. ¤

For a given schema graph define a partition-forest as a collection of partition-trees such that each node of the schema-graph appears in one and only one tree. Partition-forests will be our basis for distributing data.

Given a partition-forest F, identify those relations that are roots of the trees in F. Partition each root, and that induces a partition of each non-root node in the corresponding tree. Assigning the root fragments to sites then achieves a perfect subdivision of the database in which the fragments of subordinate nodes follow the corresponding

we choose a design set of queries, and for each choice how should the relations be partitioned?

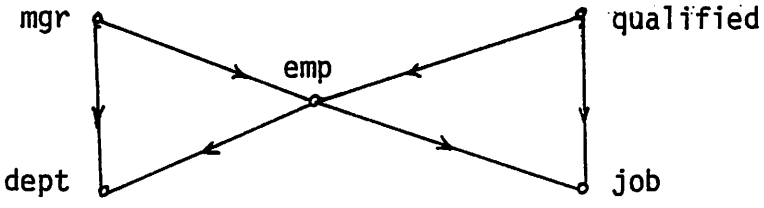
First, we note that if each relation is subdivided horizontally then every one-variable query is locally sufficient. It seems reasonable that in most situations one would want all one-variable queries to be in the design set. Hence, we assume that each relation is always subdivided horizontally.

Next, we introduce a graph representation. A relational schema designed according to the rules of Section 3 can be represented as a directed graph (called schema-graph) as follows:

(a) The nodes are in one-to-one correspondence with the relations of the schema.

(b) The arcs are in one-to-one correspondence with the foreign keys such that, if a foreign key domain in R is the primary key in S then the arc points from R to S.

Example 5.1 The Schema of Example 3.2 is Represented by the Following Graph:



Observe that an arc: R→S represents a function mapping R into S, and R is partitioned into disjoint subsets by the values of the primary key of S. Thus, for example emp $\xrightarrow{\text{assign-dno}}$ dept partitions emp into subsets each corresponding to a different department. It follows, therefore, that any partition of S yeields a partition of R via a function R→S.

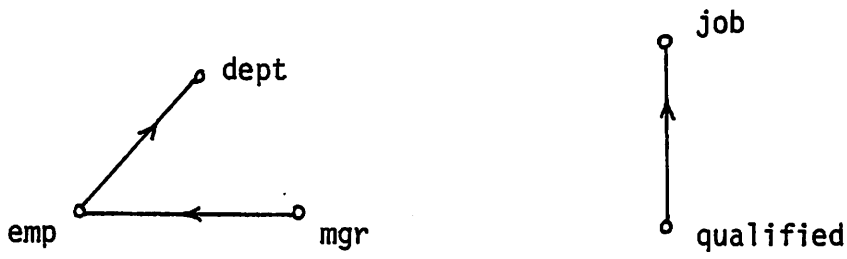
fragments of the roots. We call such a subdivision an F-induced subdivision. Our first procedure for designing a distributed database is simply the following.

(a) Given a schema graph, find a partition-forest F.

(b) Find an F-induced subdivision of the database, and identify it as the materialization M of the distributed database.

Given an F-induced subdivision M, the class Q of locally sufficient queries can be determined from F. We shall now present a way of doing so. Let R be a relation on the schema. An identifier domain in R is said to be F-supported if it corresponds to an arc in a tree in F. Consider the following example based on the schema of Example 5.1:

Example 5.3 Let F be given by:



The F-supported domain are

dno in emp and dept

eno in mgr and emp

jid in qualified and job

Relations produced by relational-algebraic operation inherit the F-supported domains. Thus, for example, dno is an F-supported domain of emp $\overset{eno}{\bowtie}$ mgr because it is one for emp.

Proposition 5.2. Let F be a partition-forest. Let Q(F) denote the smallest class of queries such that:

(a) Q(F) includes all one-variable queries on the base relation (i.e., the relations specified in the schema).

(b) Q(F) is closed under projection, restriction, and join on an F-supported domain.

Then, an irredundant materialization M is locally sufficient with respect to Q(F) if and only if M is F-induced.

Remark: M, being irredundant, is also minimally redundant with respect to Q(F).

Proof: First, we prove that an F-induced materialization M(F) is Q(F) locally sufficient. Let Q_n denote the subset of Q(F) involving n or fewer joins, and let $R(Q_n)$ denote the set of relations obtained by Q_n acting on the database. Every identifier domain D is partitioned by M(F) into disjoint subsets D_i with D_i corresponding to site i.

Let $R \in R(Q_n)$. We shall prove by induction that:

(a) $R = \bigcup_i R_i$ where each R_i is produced by local operations, and

(b) if D is an F-supported domain in R then $R_i[D_j] = \phi$ for $i \neq j$ where $[D_j]$ denotes restriction on D_j.

First, consider $R \in R(Q_0)$. Then R must be of the form

$$R = \pi\rho B$$

when B is one of the base relations, π is a projection, and ρ a restriction. B is partitioned by M(F), so that

$$R = \pi\rho \sum_i B_i = \pi \sum_i \rho B_i$$

$$= \bigcup_i \pi\rho B_i$$

Now, if D is F-supported in R then it must also be F-supported in B and

$B_i = B[D_i]$. Hence,

$$R_i[D_j] = \pi\rho(B[D_i])[D_j]$$
$$= \pi\rho B[D_i \cap D_j] = \phi$$

and properties (a) and (b) are proved for n = 0.

Assume (a) and (b) to be true for $n \leq m$, and consider $R \in R(Q_{m+1})$. R is expressible as

$$R = \pi\rho(A \overset{D}{\bowtie} B)$$

where $\overset{D}{\bowtie}$ denotes joint on D, and A and B are in $R(Q_m)$ with D as an R-supported domain. Now,

$$A \overset{D}{\bowtie} B = \sum_i A[D_i] \overset{D}{\bowtie} B[D_i]$$

and with property (b) we have

$$A \overset{D}{\bowtie} B = \sum_i A_i[D_i] \overset{D}{\bowtie} B_i[D_i]$$

Since A_i and B_i are locally generated, so is $A_i[D_i] \overset{D}{\bowtie} B_i[D_i]$. Therefore,

$$R = \pi\rho \sum_i A_i[D_i] \overset{D}{\bowtie} B_i[D_i]$$
$$= \bigcup_i \pi\rho(A_i[D_i] \overset{D}{\bowtie} B_i[D_i])$$
$$= \bigcup_i R_i$$

and (a) is proved.

For (b) let D_i denote an F-supported domain in R. Clearly, D_i must be an F-supported domain in at least one of the pair (A,B). With no loss of generality assume D_i to be supported in A. Then,

$$A_i[D'_j] = \phi$$

and

$$R_i[D'_j] = \pi\rho((A_i[D'_j])[D_i] \overset{D}{\bowtie} B_i[D_i])$$

$$= \phi$$

so that (b) is proved. By induction, properties (a) and (b) are true for all n.

We note that $q \in Q(F)$ implies $q \in Q_n$ for some n and the "if" part of Proposition 5.2 is proved. Next, we shall prove the "only if" part by showing that any $Q(F)$-locally-sufficient M must be F-induced.

Let R be a relation corresponding to any node in F. Then there is a unique path from R to a root relation $R(0)$.

$$R = R(n) \rightarrow R(n-1) \rightarrow \ldots \rightarrow R(0)$$

where each link ($\rightarrow$) corresponds to an F-supported domain. It follows that the join:

$$R(n) \bowtie R(n-1) \ldots \bowtie R(0) = J$$

is in $Q(F)$ and so is $\pi(J|R)$, the projection of J on the domains of $R = R(n)$. Since M is irredundant it must partition R(0). Because M is locally sufficient with respect to $\pi(J|R)$ the fragment of R at site i is obtained from the ith fragment of R(0) by the operation $\pi(J|R)$. This is exactly how an F-induced materialization was defined. The proof is now complete. ¤

6. Update Through Local Operations

The price to be paid for achieving a grater degree of local sufficiency is increased update complexity. This is true even when the materialization inovlves no redundancy. Basically, this is because

local sufficiency for Q(F) is achieved only when the distribution of data (i.e., M) satisfies the following integrity constraint:

> Every identifier domain D is partitioned by M so that if D is F-supported in R then
>
> $$R_i = R[D_i]$$
>
> where R is any base relation and R_i is the fragment of R in M_i.

For example, in a materialization induced by the F of Example 5.3, fragments of "emp" are determined by dno, "mgr" by eno, and "qualified" by jid. On updates the integrity constraint $R_i = R[D_i]$ must be verified for each relation that is affected.

In a distributed system the communication cost on updates has two components: message traffic and synchronization delay. If every update can be accomplished by broadcast then the synchronization delay, at least, is minimized. In this section we attempt to isolate those updates that can be effected by broadcast and propose an update protocol to take advantage of this property.

We define an update to be <u>locally realizable</u> if it can be completed by broadcasting and updating in place. For example, if every base relation is horizontally subdivided then any one-variable deletion operation (e.g., delete e where e. age > 65) is locally realizable.

The insertion of a single tuple is also locally realizable, but somewhat more complex. We note that, first, the materialization being irredundant, insertion is done at only a single site; and second, there is a difference between inserting in a relation that corresponds to a root node in F and one that does not. For a root insertion, there has

to be an algorithm for allocating a new tuple to a specific site. The tuple is then sent to the designated site, or the tuple and its site designation are broadcast. To insert a tuple in a non-root relations, the tuple is broadcast and upon its reception, each site checks the F-supported foreign key (there is only one) value in the received tuple. The tuple is installed only at the site (again, there is only one) that hosts that key value. For example, assume that M is induced by the F of Example 5.3, and that the tuple:

e = (eno = 12345, ename = "F. Fox," age = 32, dno = 37, jid = 213)

is to be inserted in emp. The domain dno is F-supported in emp. On receiving the broadcast instruction to insert e, each site must check on the existence of "dno = 37" and e is inserted at the only site where the existence is verified. If "dno = 37" represents a new department then the appropriate insertion to dept must precede the insertion of e.

Changing values in a tuple is also locally realizable except for changes to the primary key or any F-supported foreign keys. Changing primary keys can be assumed to be a prohibited operation as it is in most systems. Changing an F-supported foreign key is potentially non-locally-realizable. Consider, for example, changing the department to which a given employee is assigned. For the F in Example 5.3 "employees" follow "departments" in the assignment of data to sites. Changing the department may well require a tuple in emp to migrate from one site to another, and is thus not locally realizable. This difficulty is circumvented by requiring an update to an F-supported foreign key to be effected by a pair of depletion-insertion operations. For example, changing dno from

37 to 12 for the employee with "eno = 12345" would require the following pair of operations:

delete e where e.eno = 12345

append to emp(eno = 12345, name = "F.Fox,"
age = 32, dno = 12, jid = 213)

In summary, all one-variable updates are locally realizable except changes to primary and F-supported keys. The former is an operation that should be prohibited, and the latter must be replaced by a deletion-insertion pair if all updates are to be locally realizable.

7. Replicating Data to Enhance Retrieval

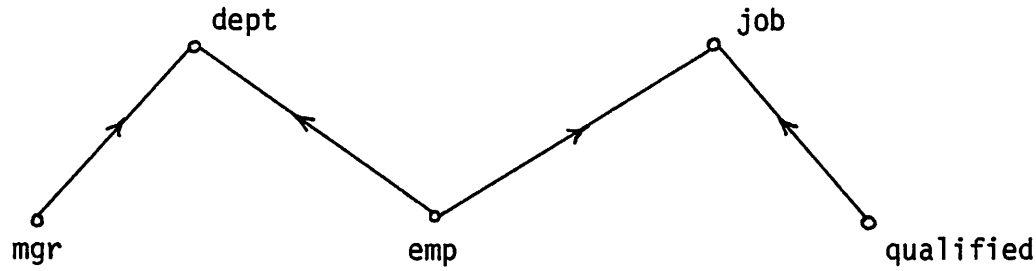
Without data replication, about the best one can do in distributing data is to pick a "partition forest" F from a schema graph and use it to induce a materialization M. In general, F being only a subgraph of the schema graph, there are semantic queries for which M is not locally sufficient. Another view of this problem is afforded by noting that any collection of semantic queries corresponds to a subset of the schema graph, and if the subset is not a partition forest, then an irredundant materialization cannot be locally sufficient for the entire collection.

Example 7.1. Suppose that for the schema graph in Example 5.1 we require M to be locally sufficient for the following joins:

dept $\rhd\!\lhd$ mgr dept $\rhd\!\lhd$ emp

job $\rhd\!\lhd$ emp job $\rhd\!\lhd$ qualification

The subgraph to support them is given as follow:



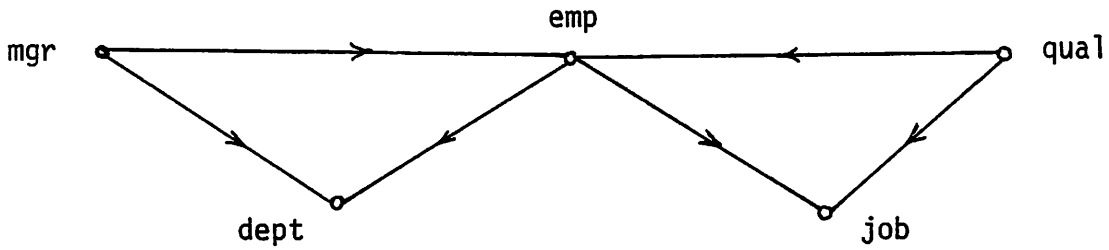
This not being a partition forest, there exists no irredundant materialization that is locally sufficient for all four of the specified joins, except for the trivial materialization of having all data at a single site. However, there are many ways of replicating data to obtain a locally sufficient materialization.

Replicating data to improve performance is hardly a new idea. In the context of distributed processing, it has been considered as a tactic of optimizing file allocation [APER 80, CHU 73, MAHO 76]. Our approach is significantly different in that we exploit the semantic information in a database schema in deciding how to replicate data. Three approaches to replicating data are discussed here: denormalization, all-or-none, and multiple-partitions.

The idea underlying denormalization is exceedingly simple. For a given materialization M, call a query unsupported if M is not locally sufficient. The idea is to preprocess any unsupported query and add it to the database before considering the problem of partitioning and distributing the database. The following example illustrates the procedure.

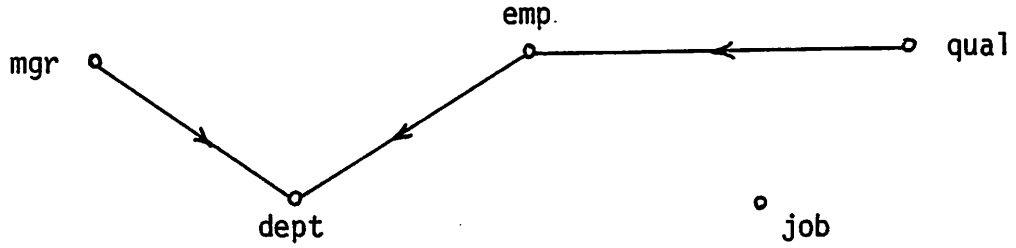
Example 7.2

Consider the schema graph for the "Company" database considered in Example 5.1:



A possible partitioning forest is:

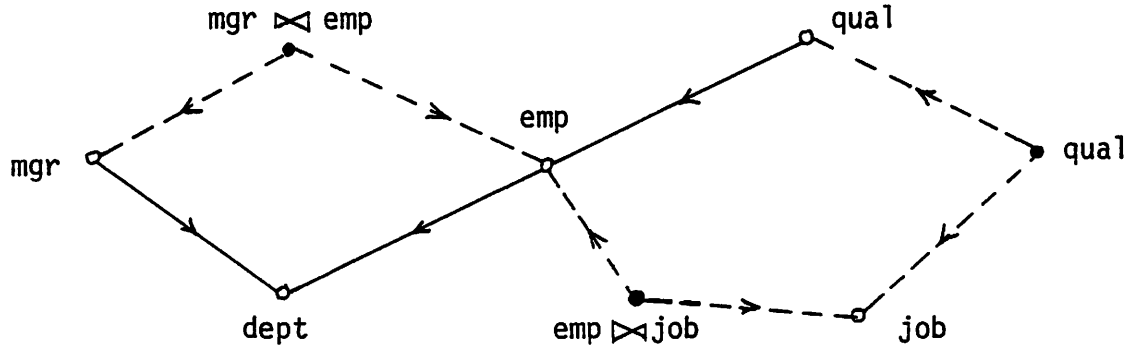
Partitioning Forest



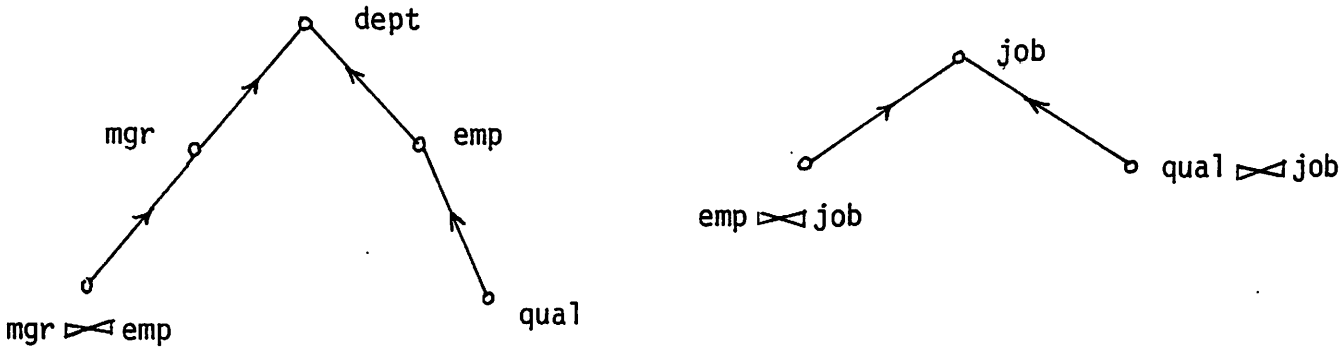
The mgr→dept, emp→dept, and qual→emp semantic joins are supported by this partitioning, while mgr→emp, emp→job, and qual→job are not. Queries involving the latter three joins are not locally sufficient.

Suppose that we add the joins: mgr ⋈ emp, emp ⋈ job, qual ⋈ job to the database and show these as new nodes in the schema graph with arcs directed to the relations that participate in the corresponding joins.

Denormalization



Now extract the following partition forest from the denormalized schema graph and note that it contains the previous partition forest:



In this forest, all natural joins involving any two relations from the original schema are supported, but not all semantic queries. For example, emp ⋈ qual ⋈ job is not supported.

Partitioning and distributing the denormalized (hence replicated) database can now proceed as before using the procedure given in Section 5. In practice, it is probably desirable to distribute semijoins rather than joins. For example, once we have partitioned emp ⋈ job into disjoint fragments $(\text{emp} \bowtie \text{job})_i$, we can project $(\text{emp} \bowtie \text{job})_i$ on the domain of emp and job respectively to get fragments of emp and job, which can then be grouped with the fragments obtained from partitioning emp and job directly. In this way, only fragments of the original relations need be stored locally.

Denormalization certainly increases parallelism for retrievals, but at the price of making updates more difficult. For each tuple the number of replicated copies and where they reside may be difficult to determine, and on update tuples may have to migrate. All in all, denormalization is probably not a good idea for dynamic data.

A better way of replicating dynamic data is "all-or-none." Here, each relation is either partitioned into non-overlapping fragments, or it is fully replicated at every site. Clearly, updating such a distributed database is no more complex than updating an irredundant one. The question is: how do we decide which ones to replicate?

Consider any subgraph G of the schema graph, including the schema graph itself. A node x on G is said to be nonconflicting (G) if:

(a) no arc emanates from x, or

(b) exactly one arc emanates from x and is directed to a nonconflicting node.

In all other cases, x is said to be conflicting. For example, in the schema graph of Example 5.1 (also shown in Example 7.2) dept and job are nonconflicting nodes, and the other three are conflicting. In Example 7.1, emp is the only conflicting node.

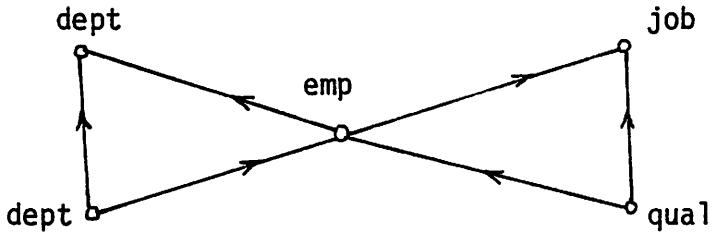
A procedure for constructing a materialization with all or none replication is simply the following. Replicate the conflicting nodes and partion the nonconflicting ones. The reason why this works is that the subgraph of G that contains only the nonconflicting nodes is always a partition forest.

A materialization with many fully replicated relations is of dubious value. One achieves parallelism in such a situation, but the parallel efforts are duplicative. For example, fully replicating every relation enjoys no advantage in parallelism over placing the entire database at a single site. (It may enjoy an advantage in communication.) We shall consider an approach, "multiple partitions," that combines features from both denormalization and all-or-none.

The idea underlying "multiple partitions" is to replicate the conflicting nodes in a graph while keeping the number of arcs the same, so as to eventually produce a partition forest. We shall illustrate the idea with an example, but omit any proof for the general case.

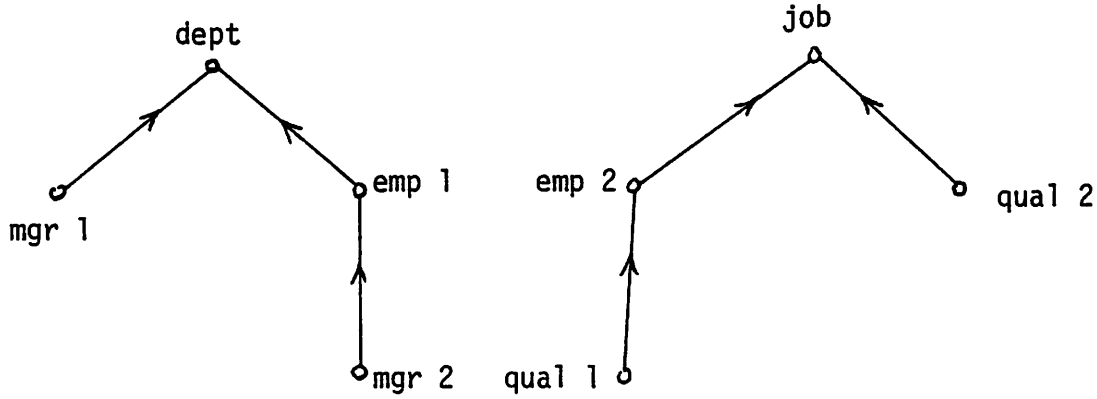
Example 7.3

Consider the schema graph in Example 7.2:



Duplicate the conflicting node emp, and let each arc connected to emp be connected to one or the other, but not both, of the replicas.

Now, only mgr and qual are condlicting, and we can replicate each to get a partition forest:



Any partition of dept and job induces a perfect partition of each copy of emp, mgr, and qual. At each site, the fragments from different copies of the same relation can be merged if desired. However, for each tuple at each site we need to know which copies contain it. Compared to all-or-

none replication, replication by multiple-partitons incurs less redundancy but greater complexity on updates.

8. Conclusion

To attain a high degree of parallelism in a distributed database, one has to distribute the data in a way as to minimize the need for moving data between sites. In this paper we advance the thesis that doing so requires semantic information concerning the data.

To make precise the notion of parllelism without commiting to a quantitative cost function, we introduce the notion of "local sufficiency" for distributed databases. Given a database schema with a limited amount of semantic information, we demonstrated a procedure to find those classes of queries for which local sufficiency without replication can be achieved. For classes of queries that require replication to achieve local sufficiency, three approaches to replication are proposed. Each enjoys a different blend of cost and benefit.

Acknowledgement

Research sponsored by the Air Force Office of Scientific Research grant AFOSR-79-3596, the National Science Foundation grant ECS-8007684, and the Office of Naval Research contract N00014-80-C-0507.

References

[APER 80] Apers, P.M.G., "Redundant Allocation of Relations in a Communications Network," Proc. Fifth Berkeley Workshop on Distributed Data Management and Computer Networks (Feb. 1981).

[CHEN 76] Chen, P.P., "The Entity-Relationship Model-Toward a Unified View of Data," ACM Trans. Database Systems, V1, N1 (Mar. 1976).

[CHU 73] Chu, W.W., "Optimal Allocation of Files in Computer Networks," in Computer Communications Networks, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[MAHO 76] Mahoud, S., J.S. Riordan, "Optimal Allocation of Resources in Distributed Information Networks," ACM Trans. on Database Systems, V1, N1 (Mar. 1976).

[WONG 79] Wong, E., Katz, R.H., "Logical Design and Schema Conversion for Relational and DBTG Databases," Proc. Intl. Conference on Entity-Relationship Approach to Systems Analysis and Design (Dec. 1979).

[WONG 81] Wong, E., "Dynamic Re-Materialization: Processing Distributed Queries Using Redundant Data," Proc. 5th Berkeley Workshop on Distributed Data Management and Computer Networks (Feb. 1981).