tion in distributed databases," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 195–202, May 1979.
[10] P. M. Merlin, "A methodology for the design and implementation of communication protocols," *IEEE Trans. Commun.*, vol. COM-24, pp. 614–621, 1976.
[10] J. B. Rothnie, Jr. and N. Goodman, "A survey of research and development in distributed database management," in *Proc. IEEE 3rd Int. Conf. Very Large Databases*, 1977.
[11] D. Skeen, "Nonblocking commit protocols," presented at the SIGMOD Int. Conf. Management of Data, Ann Arbor, MI, 1981.
[12] —, "Crash recovery in a distributed database management system," Ph.D. dissertation, Dep. Elec. Eng. Comput. Sci., Univ. Calif., Berkeley, 1982.
[13] M. Stonebraker, "Concurrency control and consistency of multiple copies in distributed INGRES," *IEEE Trans. Software Eng.*, vol. SE-5, May 1979.
[14] R. Schapiro and R. Millstein, "Failure recovery in a distributed database system," in *Proc. 1978 COMPCON Conf.*, Sept. 1978.
[15] L. Svobodova, "Reliability issues in distributed information processing systems," in *Proc. 9th IEEE Fault Tolerant Comput. Conf.*, Madison, WI, June 1979.

Dale Skeen was born in Concord, NC. He received the B.S. degree in computer science from North Carolina State University, Raleigh, and the Ph.D. degree in computer science from the University of California, Berkeley, in 1982.

He is currently an Assistant Professor in the Department of Computer Science, Cornell University, Ithaca, NY, a position he has held since 1981. His current research interests include distributed databases, highly fault-tolerant distributed systems, parallel algorithms for CAD/VLSI.

Dr. Skeen is a member of the Association for Computing Machinery, the IEEE Computer Society, and Phi Kappa Phi.

Michael Stonebraker, photograph and biography unavailable at the time of publication.

Dynamic Rematerialization: Processing Distributed Queries Using Redundant Data

EUGENE WONG, FELLOW, IEEE

Abstract—**In this paper an approach to processing distributed queries that makes explicit use of redundant data is proposed. The basic idea is to focus on the dynamics of materialization, defined as the collection of data and partial results available for processing at any given time, as query processing proceeds. In this framework the role of data redundancy in maximizing parallelism and minimizing data movement is clarified. What results is not only the discovery of new algorithms but an improved framework for their evaluation.**

Index Terms—**Distributed databases, distributed query processing, query processing.**

I. Introduction

IN THIS PAPER we propose a new formulation for the problem of processing queries in a distributed database system. By such a system we mean a collection of autonomous processors, communicating via a general communication medium, and accessing separate and possibly overlapping fragments of a database. The user's view of data is to be an integrated whole, both fragmentation and redundancy being invisible. Geographical dispersion, although sometimes present, is not an essential ingredient of such a system, and the range of systems so encompassed includes not only the classical geographically distributed databases but also configurations that are in effect database machines. The problem of distributed execution of queries is common to all these systems.

In the query processing algorithm designed for the SDD-1 distributed database management system [6], an irredundant subset of the database is used during the execution of any single query. No effort was made to exploit the possible existence of multiple copies either to maximize parallel operations or to minimize data moves. A related and somewhat hidden characteristic inherent in the SDD-1 algorithm is that parallel processing is opportunistic rather than deliberate.

These characteristics were recognized in [2] where the emphasis fell heavily on maximizing parallelism. The algorithm proposed there, and implemented for the distributed version of INGRES, achieves a high degree of parallelism by partitioning one relation among the processing sites and replicating all other needed relations at every site. We shall call this the FR (fragment and replicate) algorithm. For a query referencing many relations, the degree of data replication and the resulting communication cost to achieve this replication may be prohibitive. Thus, the FR algorithm is best applied to pieces of a many-variable query, one at a time, each with only two or three variables. Experience of using the FR algorithm in the

Manuscript received November 11, 1980; revised October 6, 1981. This work was supported by the Corporate Computer Science Center, Honeywell Corporation, the U.S. Army Research Office under Grant DAAG29-79-C-0182, and the U.S. Air Force Office of Scientific Research under Grant 78-3596.

The author is with the Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California, Berkeley, CA 94720.

0098/5589/83/0500-0228$01.00 © 1983 IEEE

distributed version of INGRES [3] indicates that the procedure of splitting a query before applying the FR algorithm is not an easy one to optimize.

It is time then, to seek a new formulation of the problem of distributed query processing that puts the issue of redundancy and parallelism into better focus. One such formulation was suggested by some recent work on database partitioning in a distributed system [9].

II. Partitioning a Database

Let $\mathfrak{D}$ denote the database as viewed by a user. Let M_i denote the data residing at processing site i. We assume that $\cup_i M_i = \mathfrak{D}$ and call $M = \{M_i\}$ a *materialization* of $\mathfrak{D}$. Suppose that the database designer is free to choose M. How should he choose?

Among the major issues to be resolved is that of redundancy. Intuitively, the cost of redundancy is paid on updates and benefit accrued on retrieval. What we need is a conceptual framework to make this precise. Let Q denote a collection of queries on a database $\mathfrak{D}$. We shall say that a materialization M of $\mathfrak{D}$ is *locally sufficient* (relative to Q) if for every q in Q and for every i there exists a local query q_i on M_i such that

$$\text{result}(q, \mathfrak{D}) = \bigcup_i \text{result}(q_i, M_i).$$

Local-sufficiency means that no intercommunication is necessary to process q. The only data movement needed is a final one to collect the results.

For two materializations M and M' denote $M > M'$ if $M_i \supseteq M_i'$ for every i. A locally sufficient M is said to be *minimally redundant* if there exists no $M' < M$ (other than M itself) that is locally sufficient. Minimum redundancy means that data reduction at local sites cannot preserve local sufficiency.

Suppose we assume that it always takes longer to process a query when there are more data. Then, in terms of both retrieval and update, it is better to have minimal redundancy than not. Thus, minimally redundant materializations represent a desirable class of partitions for the database.

III. Query Processing by Dynamic Rematerialization

In terms of the concepts that we have introduced for database partitioning, query processing can be viewed as a dynamic process of changing materializations. Let q be a single query. Let $M_i^{(t)}$ denote the data at i available and selected for processing q at any stage t of processing. $M^{(t)} = \{M_i^{(t)}\}$ will be called the materialization at t. Any algorithm for distributed query processing can be represented as a sequence of *states:* $(q^{(t)}, M^{(t)})$, $t = 0, 1, 2, \cdots, N$. The terminal state $(q^{(N)}, M^{(N)})$ is required to be locally sufficient and to satisfy the condition

$$\bigcup_i \text{result}(q_i^{(N)}, M_i^{(N)}) = \text{result}\left(q, \bigcup_i M_i^{(0)}\right).$$

In other words, from the terminal state only local processing and gathering up of results are needed to complete processing. Transition between two successive states $(q^{(t)}, M^{(t)})$ and $(q^{(t+1)}, M^{(t+1)})$ occurs as a result of data movement and/or local processing. A transition will be called a *redistribution* if only data movement is involved, and a *local derivation* if: 1) $(q^{(t+1)}, M^{(t+1)})$ is derived from $(q^{(t)}, M^{(t)})$ by local processing and 2) result $(q^{(t+1)}, M^{(t+1)})$ = result $(q^{(t)}, M^{(N)})$.

For any terminal state $(q^{(N)}, M^{(N)})$ a measure of the parallelism that it affords is given by

$$\tau(q^{(N)}, M^{(N)}) = \max_i (\text{time to process } q_i^{(N)} \text{ on } M_i^{(N)}).$$

The cost to reach $(q^{(N)}, M^{(N)})$ can be expressed as

$$C(q^{(N)}, M^{(N)}) = C_0(N) + \sum_{t=1}^{N} T_t((q^{(t-1)}, M^{(t-1)}), (q^{(t)}, M^{(t)}))$$

where $C_0(N)$ is the cost of resynchronization between transitions and T_t is the cost of making the tth transition.

If a compatible scale for τ and C is known, the problem can then be stated as one of optimal control. Even though the optimization problem is unlikely to be solved in any general sense, it provides a framework that allows algorithms proposed on heuristic grounds to be evaluated.

IV. Strategies Based on an Initially Feasible Solution

Let $q^{(0)} = q$ be the query to be processed and $M^{(0)}$ the data initially available for processing q. We say (q, M) is an *initially feasible solution* if it is a "locally sufficient redistribution" of $(q^{(0)}, M^{(0)})$, i.e., M is locally sufficient and derivable from $M^{(0)}$ by moving data. The cost of using such a strategy consists of several components, of which we assume the following to be dominant:

a) $C(M^{(0)}, M)$ = cost of moves

b) $\tau(q, M)$ = cost of terminal parallel processing.

We shall say a strategy is of the *IFS type* if it consists of the following steps:

1) One seeks a $(q^{(1)}, M^{(1)})$ that is a "local derivation" of $(q^{(0)}, M^{(0)})$.

2) If no such local derivation can be found, one seeks an initially feasible solution (q, M).

3) One seeks to improve (q, M) by replacing the one-transition strategy $(q, M^{(0)}) \rightarrow (q, M)$ by a "short" sequence of transitions. Perform the first transition $(q, M^{(0)}) \rightarrow (q^{(1)}, M^{(1)})$ in the sequence.

4) Iterate, with $(q^{(1)}, M^{(1)})$ replacing $(q^{(0)}, M^{(0)})$.

Both the SDD-1 and FR algorithms are variations of IFS algorithms. In the SDD-1 case, the initially feasible solution (q, M) is restricted to be not merely locally sufficient but single-site sufficient. That is, there exists a site j such that q can be processed entirely on M_j. The choice for M in the FR algorithm is to replicate every relation but one, which is locally sufficient for almost all q. It seems clear that to qualify for selection as the initial choice as a feasible solution, (q, M) should be at least "noninferior" with respect to the pair of costs $(C(M^{(0)}, M), \tau(q, M))$. That is, there exists no initially feasible solution that is equal or better in both (C, τ) and strictly better in at least one. Neither the SDD-1 nor the FR algorithm guarantees this in general.

Example 4.1: Consider a conceptual schema given by

person (socsec, name, state-of-res)
corp (cid, cname, state-of-inc)
emp (socsec, cid, position, salary).

Suppose that there are two sites with their local schemas given by the following view definition statements:

range of p is person
range of c is corp
range of e is emp
define person 1 (p.all) where (p. state-of-res = "NY")
define corp 1 (c.all) where (p.state-of-inc = "NY")
define emp 1(e.all) where (e.salary $\leqslant$ 25000)
define person 2 (p.all) where (p.state-of-res $\neq$ "NY")
define corp 2 (c.all) where (c.state-of-inc $\neq$ "NY")
define emp 2 (e.all) where (e.salary $>$ 25000).

Now consider a query

retrieve (p.name) where (p.socsec = e.socsec)
and (c.cid = e.cid)
and (c.name = "IBM").

We begin with the materialization.

$M_1^{(0)}$ = (person 1, corp 1, emp 1)

$M_2^{(0)}$ = (person 2, corp 2, emp 2).

Processing the clauses that involve only local operations, we get

$M_1^{(1)} = (P1, C1, E1)$

$M_2^{(1)} = (P2, C2, E2)$

where

Pk = person k projected on (socsec, name)

Ck = corp k restricted to (name = "IBM") and projected on (cid)

Ek = emp k projected on (socsec, cid)

Now assume the following statistics for these relations:

relation	# tuples	tuple width in bytes
$P1$	1000	29 (9, 20)
$P2$	1000	29 (9, 20)
$C1$	1	5
$C2$	0	5
$E1$	1000	14
$E2$	1000	14
$C1$ (cid) $E2$	100	14
$C1$ (cid) $E1$	1000	14

The initial feasible solution in the SDD-1 algorithm would consist of site 1 as the final processing site and

$\mathfrak{M}$ = {move $P2$ and $E2$ to site 1}

which entails moving 43 kbytes of data. On the other hand, the FR algorithm would yield a materialization

$M_1 = (P1, C1, E1, E2)$

$M_2 = (P2, C1, E1, E2)$

with

$\mathfrak{M}_1$ = {move $E2$ to site 1}

$\mathfrak{M}_2$ = {move $C1$ and $E1$ to site 2}

which in this example corresponds to the $M^{(2)}$ that minimizes communication cost $C(M)$ and entails moving 28 kbytes of data. $\mathfrak{M}_2$ can be reduced by joining $C1$ to $E1$ and moving the join instead of $E1$ (1405 bytes). $\mathfrak{M}_1$ can be reduced by moving $C1$ to site 2, joining $C1$ with $E2$, and moving the join. The resulting sequence of materialization would appear as follows, where ∞ denotes join:

$M^{(1)} = \{(P1, C1, E1), (P2, E2)\}$

$M^{(2)} = \{(P1, C1, E1), (P2, C1, E2)\}$

$M^{(3)} = \{(P1, E1 \infty C1), (P2, E2 \infty C1)\}$

$M^{(4)} = \{(P1, E1 \infty C1, E2 \infty C1), (P2, E1 \infty C1, E2 \infty C1)\}$

and $M^{(4)}$ is now locally sufficient. The total amount of data moved is 2805 bytes, and no more processing is involved than either the FR or the SDD-1 algorithm. For our example, the strategy that we have found is just about the best possible over a wide range of relative costs for communication and local processing.

V. Repeated-Join Strategies

The database-partition problem suggests the following class of query processing strategies: Consider a relational database $\mathfrak{D} = \{R_1, R_2, \cdots, R_m\}$ where R_k are relations. We shall say a query q is *admissible* if it is a finite repetition of "restriction," "projection," and "join" on the relations in $\mathfrak{D}$. We shall say an admissible q is *elementary* if it involves at most one join. Now, suppose that for any $\mathfrak{D}$ we know how to find a "good" materialization $M(\mathfrak{D})$ that is locally sufficient for all elementary queries. Then, we can construct a query processing algorithm as follows: construct a sequence

$$\mathfrak{D} = \mathfrak{D}^{(0)}, \mathfrak{D}^{(1)}, \cdots, \mathfrak{D}^{(N)}$$
$$q^{(0)}, q^{(1)}, \cdots, q^{(N)}$$

such that $q^{(N)} = q$, and for each t $q^{(t)}$ is an elementary query on $\mathfrak{D}^{(t)}$ and $\mathfrak{D}^{(t+1)} \subset \{\mathfrak{D}^{(t)}, \text{result } (q^{(t)}, \mathfrak{D}^{(t)})\}$. Since for each $\mathfrak{D}^{(t)}$ we know how to find a materialization $M(\mathfrak{D}^{(t)})$ that is locally sufficient for all elementary queries, $M(\mathfrak{D}^{(t)})$ is *a fortiori* locally sufficient for $q^{(t)}$. The *repeated-join* algorithm consists of repeating for each t the following steps.

a) Execute $q^{(t)}$ on $M(\mathfrak{D}^{(t)})$.

b) To obtain $\mathfrak{D}^{(t+1)}$, add result $(q^{(t)}, \mathfrak{D}^{(t)})$ to $\mathfrak{D}^{(t)}$ and eliminate the relations no longer needed in processing q.

c) Construct $M(\mathfrak{D}^{(t+1)})$.

How good this algorithm is depends on:

1) whether we can construct $M(\mathfrak{I}^{(t)})$ as claimed, and

2) the cost in resynchronization and data movement in making the transition $M(\mathfrak{D}^{(t)}) \rightarrow M(\mathfrak{D}^{(t+1)})$.

Our preliminary study suggests that the efficacy of this class of algorithms is enhanced if we augment the semantics of the relational model and use the semantics to restrict the class of admissible queries.

Roughly speaking, the semantic augmentation that we undertake corresponds to distinguishing between entities and relationships [7], [8], but we shall define the semantics strictly in terms of the constructs of the relational model.

First, we classify the sets that serve as domains of the relations in the database into *identifier* and *value*. D is an identifier domain if and only if there is a unique relation E_D such that the elements of D are in one-to-one correspondence with the tuples of E_D. We shall say D is the *key* of E_D.

Every relation must have at least one identifier domain. A relation will be called an e-relation (entity) if it has a key, and an r-relation (relationship) otherwise.

Example 5.1:

person (* socsec, name, state-of-res)

corp (* cid, cname, state-of-inc)

emp (socsec, cid, position, salary)

where underscore indicates an identifier domain and * indicates a key. Clearly, "person" and "corp" are e-relations and "emp" is an r-relation.

Now suppose that we limit the admissible data manipulation operations to the following:

a) restriction—Boolean condition on values

b) projection

c) join—on an identifier domain D

d) closure.

Note that admissible joins are limited. For example, the join

person (state-of-res = state-of-inc) corp

would be an inadmissible operation because the join is on a non-identifier domain, but the following operation is admissible:

$$(\text{person } (\overset{\text{socsec}}{\infty}) \text{ emp } (\overset{\text{cid}}{\infty}) \text{ corp}) \text{ [state-of-res = state-of-inc]}$$

Let $\overset{D}{\propto}$ denote the semijoin operator defined in [1]. That is, $A \overset{D}{\propto} B$ is the projection on A of the join $A \overset{D}{\infty} B$. The following proposition gives a condition for local sufficiency in terms of the semijoin.

Proposition 5.1.: Let $\mathfrak{D}$ be a collection of relations. Let M be a materialization of $\mathfrak{D}$ such that to each e-relation E corresponds a unique $E(k)$ in M_k such that

$$E = \bigcup_k E(k).$$

Then, M is locally sufficient for all elementary queries if for every $R \in \mathfrak{D}$ and every identifier domain D in R

$$R \overset{D}{\propto} E_D(k) \in \text{closure } (M_k) \text{ for every } k. \tag{5.1}$$

Proof: For R and S in $\mathfrak{D}$, we can write

$$R \overset{D}{\infty} S = \bigcup_k (R \overset{D}{\propto} E_D(k) \overset{D}{\infty} (S \overset{D}{\propto} E_D(k)).$$

Since projection and restriction commute with union, the proposition is proved. □

Condition (5.1) provides a simple means for testing the local sufficiency of a materialization M for elementary queries. Further, if M fails the test, (5.1) provides a means for augmenting M to make it locally sufficient. As such, (5.1) makes the repeated-join algorithm work. At each step t in the algorithm, to construct $M(\mathfrak{D}^{(t+1)})$, we only need to distribute enough of the result $(q^{(t)}, \mathfrak{D}^{(t)})$ so that

$$\text{result } (q^{(t)}, \mathfrak{D}^{(t)}) \overset{D}{\propto} E_D(k) \in \text{closure } (M_k(\mathfrak{D}^{(t+1)}) \tag{5.2}$$

for every k and every D.

Example 5.2: Take the schema in Example 5.1, and consider the same query as in Example 4.1. We have

$$\mathfrak{D} = \{\text{person, corp, emp}\}$$

$$q = \text{person } \overset{\text{socsec}}{\propto} (\text{emp } \overset{\text{cid}}{\propto} (\text{corp(name="IBM")})) \text{ [name]}.$$

Define person k and corp k as in Example 4.1, but define

$$\text{emp } k = (\text{emp } \overset{\text{socsec}}{\propto} \text{ person } k) \bigcup (\text{emp } \overset{\text{cid}}{\propto} \text{ corp } k)$$

Take the initial materialization to be

$$M^{(0)} = \{(\text{person } k, \text{corp } k, \text{emp } k), k = 1, 2\}$$

Then $M^{(0)}$ is locally sufficient for all elementary queries on $\mathfrak{D}$.

Now, take

$$\mathfrak{D}^{(0)} = \mathfrak{D} = \{\text{person, corp, emp}\}$$

$$q^{(0)} = (\text{emp } \overset{\text{cid}}{\propto} (\text{corp(name="IBM")})) \text{ [socsec]}$$

Here, we have no need to distinguish $q^{(0)}$ from its result. Hence, we can write

$$\mathfrak{D}^{(1)} = \{\text{person}, q^{(0)}\}$$

$$q = q^{(1)} = (\text{emp } \overset{\text{cid}}{\propto} q^{(0)}) \text{ [name]}$$

As in Example 4.1, assume that

$$\text{corp 2 (name="IBM")} = \phi$$

Hence, $q_2^{(0)} = \phi$ and

$$q^{(0)} = q_1^{(0)}.$$

To satisfy (5.1), we can take

$$M_1^{(1)} = (\text{person } 1, q_1^{(0)})$$

$$M_2^{(1)} = (\text{person } 2, q_1^{(0)})$$

which requires moving $q_1^{(0)}$ to site 2. Alternatively, we can take

$$M_2^{(1)} = (\text{person } 2, q_1^{(0)} \overset{\text{socsec}}{\propto} \text{person } 2)$$

which would entail first moving (person 2) [socsec] to site 1 and then moving $q_1^{(0)} \overset{\text{socsec}}{\propto}$ person 2 to site 2. However, the double move would be obviated by storing at each site an index for the distribution of identifiers.

For a given q, the sequence $q^{(n)}$ is by no means unique, and the optimization problem is to choose $q^{(n)}$ so as to minimize cost, however cost is defined.

VI. Conclusion

In this paper we propose a new approach to distributed query processing. This approach focuses on how the data available at each site change as processing proceeds. We believe that issues of parallelism and redundancy are rendered clearer by this approach. Our immediate goal is not so much to find better algorithms, but to provide a conceptual framework in which new classes of algorithms can be formulated in a natural way.

References

[1] P. A. Bernstein and D. W. Chiu, "Using semi-joins to solve relational queries," Computer Corp. Amer., Tech. Rep. CCA-01-79; to appear in *J. Ass. Comput. Mach.*

[2] R. Epstein, M. Stonebraker, and E. Wong, "Distributed query processing in a relational data base system," in *Proc. 1978 ACM-SIGMOD Conf. Management of Data*, Austin, TX, June 1978.

[3] R. Epstein and M. Stonebraker, "Analysis of distributed data base processing strategies," presented at the Int. Conf. Very Large Data Bases, Montreal, Oct. 1980.

[4] G. D. Held, M. R. Stonebraker, and E. Wong, "INGRES–A relational data base system," in *Proc. NCC*, vol. 44, 1975.

[5] A. Hevner and S. B. Yao, "Query processing on a distributed data base," in *Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks*, LBL-7953 UC-32, Lawrence Berkeley Lab., Berkeley, CA, Aug. 1978.

[6] E. Wong, "Retrieving dispersed data from SDD-1: A system for distributed databases," presented at the 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, Lawrence Berkeley Lab., May 1977.

[7] P. P. Chen, "The entity-relational model–Towards a unified view of data," *ACM Trans. Database Syst.*, vol. 1, pp. 9–36, Mar. 1976.

[8] E. Wong and R. H. Katz, "Logical design and schema conversion for relational and DBTG databases," in *Int. Conf. Entity-Relationship Approach to Systems Analysis and Design*, Los Angeles, 1980, North-Holland.

Eugene Wong (F'74) received the B.S. and Ph.D. degrees from Princeton University, Princeton, NJ, in 1955 and 1959, respectively.

After a period of service in IBM Research (1955–1956, 1960–1962), he has been on the Faculty of the University of California, Berkeley, since 1962, where he is currently a Professor of Electrical Engineering and Computer Sciences. A codesigner of the INGRES database management system and an early participant in the design of the SDD-1 distributed database system, he has been active on a number of problems in database management. These include: query processing, database design, distributed databases, schema, and program conversion. He has served on the Editorial Board of the *ACM Transactions on Database Systems* and as one of its Associate Editors.

Professor Wong is a member of the Association for Computing Machinery. He was an NSF Postdoctoral Fellow in 1959–1960, a Guggenheim Fellow in 1968–1969, and a Vinton Hayes Fellow at Harvard in 1976–1977.