SEPTEMBER 1982 VOL. 5 NO. 3

a quarterly bulletin of the IEEE computer society technical committee on

Database Engineering

Contents

Database Engineering Bulletin is a quarterly publication of the IEEE Computer Society Technical Committee on Database Engineering. Its scope of interest includes: data structures and models, access strategies, access control techniques, database architecture, database machines, intelligent front ends, mass storage for very large databases, distributed database systems and techniques, database software design and implementation, database utilities, database security and related areas.

Contribution to the Bulletin is hereby solicited. News items, letters, technical papers, book reviews, meeting previews, summaries, case studies, etc., should be sent to the Editor. All letters to the Editor will be considered for publication unless accompanied by a request to the contrary. Technical papers are unrefereed.

Opinions expressed in contributions are those of the individual author rather than the official position of the TC on Database Engineering, the IEEE Computer Society, or organizations with which the author may be affiliated.

Membership in Database Engineering Technical Committee is open to IEEE Computer Society members, student members, and associate members. (Application form in this issue.)

Letter from the Guest Editor

Query Optimization is the topic of this special issue of Database Engineering. Thirteen short papers from academic and industrial researchers give an overview of current optimization research and state-of-the-art optimizer implementations.

A number of trends are evident. Queries are becoming more varied and complex, with high-level queries, nested queries, aggregate queries, and groups of queries being considered. Target environments are also becoming diverse and complicated, ranging from centralized mainframes to distributed systems and database machine backends. Often, multiple data models are present, both as user views and in actual storage structures.

In response, optimizers are getting smarter. They are searching larger strategy spaces which include many sophisticated strategies, modeling costs more accurately, and detecting opportunities for common processing over batches of queries. In distributed systems, their functionality may be distributed. Overall, there is a better understanding of the internal structure of the query compilation process.

The first three papers deal with optimizers in centralized environments. The next five discuss distributed environments. Hevner and Yao's paper also touches on database machine backends and join processing models and hardware. Then come two papers on optimization of multiple queries. Kim's paper also comments on main memory buffer space and nested queries. The next paper covers uniformity assumptions in cost analysis and batching queries, and is followed by a paper on aggregate processing. The last paper presents a model of a physical target machine for optimization, and compares the strategy spaces of various optimizers.

One major omission is a useful tool for testing optimizers and evaluating their performance -- a benchmark set of database schemas and a sample relational query load against them, perhaps with associated population statistics. I think many researchers would appreciate seeing this kind of information published, perhaps in Database Engineering.

My thanks to the contributors to this issue for their enthusiasm, good ideas, and hard work, and to Won Kim for his support and helpful suggestions.

David S. Reiner

David S. Reiner

QUERY OPTIMIZATION IN INGRES

Robert Kooi* and Derek Frankforth
Relational Technology, Inc.
2855 Telegraph Ave., Suite 515
Berkeley, California 94075
(415) 845-1700

1. Introduction

Relational Technology Inc., which markets the INGRES relational database management system, is devoting a significant effort to query optimization. Our research is focused on two areas, the I/O subsystem and strategies for processing multi-variable queries. The goal of the I/O subsystem is to minimize the time required to perform simple repetitive queries such as those that might be found in a transaction processing environment. For multi-variable queries we are developing techniques that will evaluate a large set of query execution plans to find the least expensive strategy.

2. I/O Subsystem

The I/O system of INGRES is being changed to reflect the needs of transaction processing type applications without significantly impacting the performance of large queries. The major component of the new I/O system is a special process called the kernel process (KP). The duties of the KP are as follows:

1) Maintain all locks, and handle all multi-process synchronization.
2) Manage a global page cache that is implemented in shared memory.
3) Manage a cache of open relations and their descriptions.
4) Detect and resolve deadlocks. (planned)
5) Coordinate transactions with the system log. (planned)

The KP allows INGRES to set the locking strategy based on the expected number of I/O's that will be needed to complete the query on a per relation basis. This allows page level locking for single record transactions, and for multi-relation queries where the optimizer expects a low percentage of the relation to be scanned. Relation level locking can be requested in the cases where the whole relation will be scanned. In this case the associated locking and context switching to the KP is eliminated by opening up the relation in the INGRES process and only allowing read access to all other users. When page level locking has been requested and the KP notices logical sequential read requests, it tries to start the next read before it returns the current page, so that I/O delay time can be used to scan the current page, thus minimizing realtime response.

When an open relation request is sent to the KP, it first checks to see if page level locking has been requested. If so, the KP will handle all the I/O requests for this relation. If the relation was used recently it is probably open, and some pages that belong to it might also be in the page cache. The

---

* Some of the research reported here was originally conducted for a Ph.D. dissertation by Robert Kooi, which was supported in part by NIH USPHS Grants 5M01-RR00210 and 1P50 HD 11089 and NIAAA Grant AA03282.

page cache is a section of memory that is mapped into the address space of every INGRES process. The KP reads pages into the cache and then passes a pointer to the page to the requesting process. This gets rid of the overhead of copying pages around, which is one of the most expensive operations in CPU time at this low level in the system. The cache logic gives preference to system catalog pages, then to ISAM index pages, then lastly to regular data pages. This part can be easily changed when new storage structures are added, such as B-tree, and when system catalog accesses decrease when other planned changes in INGRES are completed.

The expected gains on single record queries can be broken down as follows. There is a cache of macro queries that are stored in parsed form and can be executed, which result in CPU time savings of 45%. The relation descriptor cache, and KP relation & page cache will cut the CPU overhead by 10-15% and the realtime delay by 50%. The ability of the cache to delay writing dirty pages, will cut the realtime delay by another 10%. So the expected CPU time savings are in the 65-70% range, while the realtime response time will be decreased by 60%. The cost of the KP is expected to be 5-10% in CPU time. This translates into a 2-3 times performance improvement for single record transactions.

More changes are planned to enhance the overall performance of INGRES. The additional expected performance increases for single record queries will amount to about 15% CPU time improvement. These changes include: a dynamic query compiler, enhanced user process communcations, and code reorganization to allow precompiled queries to be saved and quickly activated.

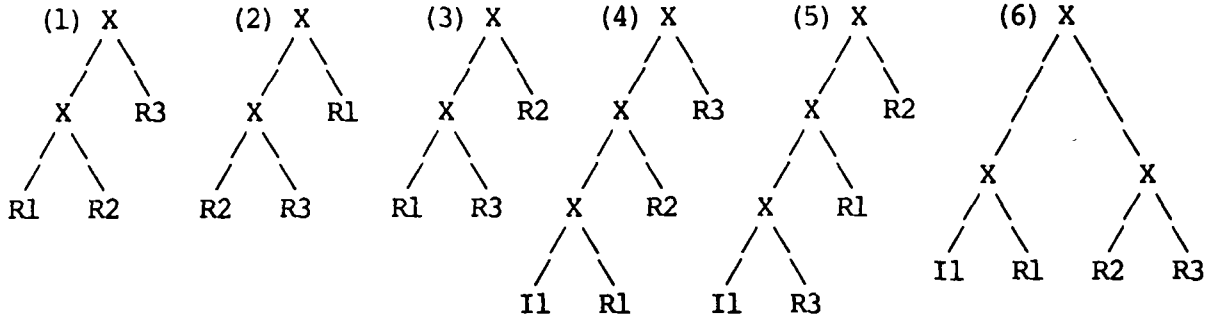
3. Multi-Variable Query Optimization

We have developed a system that can estimate, for an arbitrary query, the cost of a very large collection of query execution plans (QEP). The QEP search space includes the following:

1) N-variable, equi-join, queries.
2) Reformatting (to hash, isam or sort) of relations.
3) Generalized index usage.
4) ISAM, hash, sort-merge and tid joins.

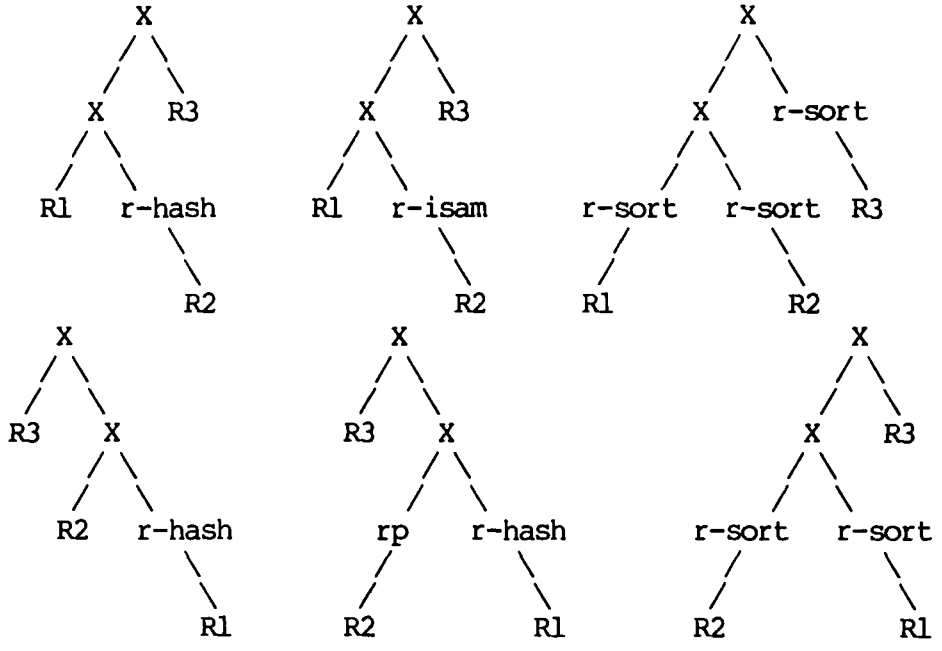
We model QEPs as binary trees where each node can be one of four operator types: Join, Restrict-Project, Reformat and Disk-Resident-Scan. The leaves of the binary tree represent relations or indexes (which are stored as relations) and correspond to the Disk-Resident-Scan operator. Interior nodes with two sons represent a Join operator, nodes with one son represent a Restrict-Project or Reformat operator. This model can represent a very large class of access strategies including most of those in [ROSE82], [WONG76], [SELI79], [BLAS76], [SMIT75] and some of those in [YAO79].

The binary trees are divided into "join" and "operator" trees. The enumeration of QEPs consists of three steps: enumeration of join trees, generation of operator trees from a given join tree and the evaluation of the cost of the operator trees. A join tree is a binary tree where each leaf node is a relation or index and each interior node is a join (with two sons). For example, the join trees for a three variable query (relations R1, R2 and R3) with one potentially useful index (I1 on R1) include (a partial list):



Notice that index usage is modeled as a join, that an index may be joined with a relation that it does not index (5) or to another index, and that the results of intermediate join trees (R2 and R3 in (2)) are saved for use in later join trees (6). N-way joins for N > 2 are not evaluated.

From a given join tree we generate operator trees by splicing in Reformat and Restrict-Project operators in all possible combinations. For example, join tree (1) would generate the following operator trees (a partial list):



Heuristics on the placement of indexes, relations and reformat operators reduce the number of possible join and operator trees, however the number of possibilities is still large. This leads to the following heuristic: If the amount of time spent on optimization so far is some fraction k of the estimated time to execute the query according to the best QEP so far, then stop optimization and use that QEP.

In an operator tree (but not in join trees) the left son of a join node represents the "outer" loop of the join and the right son represents the "inner" loop. The operation of the join operator depends on the structure of its inner and outer inputs. For instance, if the inner is hashed or ISAM then for each tuple from the outer relation, a hash or directory lookup is performed on the inner. If both inputs are sorted then a sort-merge join is performed. If the outer is an index on the inner, then a TID lookup is done.

Restrict-Project operators are applied to leaf nodes in order to reduce the amount of the relation that needs to be scanned. This happens when there are range or exact value restrictions on keys of the relation.

The cost of an operator tree is based on the estimated number of disk accesses required except for sorts where CPU time is factored in. This is calculated using statistics about the distributions of attributes which are held in systems catalogs. For each attribute in the database we have a choice of keeping nothing, minimum and maximum values, or a variable-range histogram. The accuracy of the cost determination depends on the accuracy of the statistics for the attributes. An initial implementation of this model [KOOI80] was compared to an early University of California, Berkeley version of INGRES (6.2/6) and significant improvements were found for two and more variable queries. It was found that minimum and maximum values provide a significant improvement over having no information and that queries executed according to QEPs found using histograms provided approximately a thirty percent improvement in execution time over queries executed according to QEPs found using minimum and maximum values alone. Comparisons with the current version of INGRES developed at RTI are currently under way.

4. Future Extensions

Future work on multi-variable query optimization will include extensions to handle theta-joins, multiple-attribute keys, nested queries, INGRES's aggregate functions and automatic updating of statistical information. We also plan to allow QEPs to be saved on disk to support query compilation. A byproduct of the query optimization is an estimate of the amount of time required to process the query and the size of the resulting relation. We will study how this can be used as the basis of a database design aid.

5. References

[BLAS76] Blasgen, M.W. and K.P. Eswaran, "On the Evaluation of Queries in a Relational Data Base System," Report RJ 1745, IBM Research, Yorktown Heights, New York, April 1976.

[KOOI80] Kooi, R.P., "The Optimization of Queries in Relational Databases," Ph.D. Dissertation, Case Western Reserve University, Sept. 1980.

[ROSE82] Rosenthal, A. and D. Reiner, "An Architecture for Query Optimization," Proceedings of ACM-SIGMOD, 1982.

[SELI79] Selinger, P.G., M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, T.G. Price, "Access Path Selection in a Relational Database Management System", Proceedings of ACM-SIGMOD, 1979.

[SMIT75] Smith, J.M. and P.Y.T. Chang, "Optimizing the Performance of a Relational Algebra Database Interface," Communications of the ACM, Vol. 18, No. 10, Oct. 1975.

[WONG76] Wong, E. and K. Youssefi, "Decomposition - A Strategy for Query Processing," TODS, Vol 1., No. 3, Sept. 1976.

[YAO79] Yao, S.B., "Optimization of Query Evaluation Algorithms," TODS, Vol. 4, No. 2, June 1979.

QUERY PROCESSING IN UNIVERSAL RELATION SYSTEMS†

Jeffrey D. Ullman
Stanford University
415-497-1512

I. Introduction

Universal relation systems are designed to present the user with a view of data that is a single (*universal*) relation. Just as [CODD70] proposed the relational model to free the user from navigation in the physical database, universal relation systems attempt to free the user from navigation within the conceptual database. However, because of the potential for ambiguity when the system is asked to infer the path or paths connecting attributes mentioned in a query, the successful design of universal relation systems requires some subtle mathematics and conceptual tools. Some of these ideas are described in [SAGI81], [SAGI82], [KUCK82], and [ULLM82a, b].

In this note we are concerned not so much with the technical details of supporting universal relations as with the role that a strange form of query processing, called "optimization under weak equivalence" plays in the implementation of such systems. We focus on System/U, an example of this type of facility under development at Stanford. People who have contributed to the implementation and/or underlying concepts for this project include H. Korth, G. Kuper, D. Maier, and F. Sadri.

II. Weak Equivalence

Two expressions are *weakly equivalent* if whenever the relations that appear as arguments of the expressions are the projections of a single relation that is defined over all the attributes, then the expressions produce the same answer. In contrast, the ordinary notion of equivalence, which we refer to as *strong equivalence*, requires that the two expressions yield the same result for any relations as arguments, regardless of whether the relations are the projections of a single universal relation.

Example 1: Consider a pair of relations $R(A,B)$ and $S(B,C)$, and the two expressions R and $\pi_{AB}(R \bowtie S)$‡ Let R consist of the two tuples $\{01, 23\}$, and let $S = \{14\}$. Then $R \bowtie S = \{014\}$, and $\pi_{AB}(R \bowtie S) = \{01\} \neq R$. Thus, $\pi_{AB}(R \bowtie S)$ and R are not strongly equivalent.

However, if R and S are the projections of one (universal) relation $U(A,B,C)$, then we can prove that $R \equiv \pi_{AB}(R \bowtie S)$. Intuitively, if there is a universal relation, the tuple 23 cannot appear in R unless there is a tuple $3c$ in S, for some value of c, whereupon $23c$ is in $R \bowtie S$, and 23 would appear in $\pi_{AB}(R \bowtie S)$. Thus, R and $\pi_{AB}(R \bowtie S)$ are weakly equivalent, even though they are not strongly equivalent. □

III. Tableaux

There is a convenient form for representing certain common expressions (including those built from relational algebra operations like selection, projection, and join); in this form we can provide an optimization algorithm that minimizes the number of "expensive" operations, principally joins. The idea originated with [CHAN77], but assumed its most common notational form in [AHO79a, b] and [KLUG81]. In this form, we represent expressions as in Fig. 1. On top is an (optional) header indicating the attributes that correspond to the columns. Next comes the *summary*, an indication of which symbols form the result of the expression that this *tableau* denotes. Finally come all the *rows*. Blanks in the rows are used to denote symbols that appear nowhere else.†† In some circumstances, we also append certain constraints among symbols, e.g., $a < b$.

The meaning of a tableau is defined in terms of its effect on a (universal) relation over all the attributes. Intuitively, the result of applying a tableau to some relation u is obtained by mapping the rows into the tuples of u in all possible ways that do not map the same symbol to two different symbols. For each such mapping, we list the value of the summary; the tuples so listed form the result of applying the tableau.

Example 2: Consider the effect of the tableau of Fig. 1 on the relation $\{012, 314, 567\}$. We could map both

† Work supported by AFOSR grant 80-0212.

‡ We use the relational algebra notation of [ULLM82b], where π_X stands for projection onto the set of attributes X, σ_C stands for selection by condition C, and $\bowtie$ stands for the natural join.

†† The reader may appreciate the similarity of the tableau form of expression to the form of queries in Query-by-Example (see [ULLM82b], e.g.). The summary of the tableau corresponds to placing P. in front of certain symbols in QBE.

	A	B	C
summary	a	b	
(1)	a	b	
(2)		b	c

Fig. 1. A tableau.

rows into tuple 567, which causes the summary ab to become 56. We could map the first row to 012 and the second to 314, which gives us summary 01. Note that this mapping assigns a unique value to b, namely 1, as all legal mappings must do. We cannot map the first row to 012 and the second to 567, since that would attempt to map b to both 1 and 6. If we consider all possible legal mappings and accumulate the summaries, we get the result of the tableau, $\{01, 31, 56\}$. □

There is a simple way to minimize the number of rows of a tableau while preserving weak equivalence. This minimization, while we shall not show it, has the effect of minimizing the number of joins. We reduce the tableau by looking for mappings, from all the rows into a proper subset of the rows, where the mapping has the following properties.

1. No symbol is mapped to two or more different symbols.
2. Each symbol that appears in the summary is mapped to itself.
3. Any constraints that hold on the original tableau's symbols are implied by the constraints on the symbols to which they are mapped.

If we find such a mapping, we may eliminate all rows that are not mapped onto by any row.

The above process is "finite Church-Rosser," in the sense that we can apply any such mapping we see, and still be guaranteed that after applying all the mappings we can, we shall wind up with the unique (up to renaming of symbols) minimal tableau.

Example 3: In Fig. 1, we can map both rows to row (1). This has the effect of mapping a and b to themselves, as we must by rule (2). Symbol c is mapped to the symbol represented by the blank in the first row, and the symbol represented by the blank in the second row is mapped to a.

Since no row maps to row (2), we may eliminate that row, leaving the tableau of Fig. 2. While we shall not show it, let us comment that where weak equivalence is concerned, Fig. 1 represents the expression $\pi_{AB}(R \bowtie S)$ discussed in Example 1, while Fig. 2 represents expression R. The existence of the mapping we found proves the weak equivalence of these two expressions. □

	A	B	C
	a	b	
(1)	a	b	

Fig. 2. Reduced tableau.

IV. A Universal Relation System

There is an important application of the weak equivalence idea and the reduction of tableaux that we have just described. In "System/U," a universal relation system being implemented at Stanford [ULLM82a], queries are interpreted as applying to one or more copies of a universal relation, and that universal relation is constructed by taking the natural join of all relations in the database.† However, before answering the query, we optimize this join by applying the tableau minimization algorithm discussed above. This optimization has two beneficial effects.

1. By eliminating joins, the response to the query can be obtained more quickly.
2. Far more importantly, the elimination of join terms that are extraneous to the query provides the user

† Technically, it is only for the simplest databases that all the relations are joined to form the universal relation. In complicated databases, the possibility of multiple paths among attributes is accounted for in System/U by replacing the universal relation by the union of the *maximal objects*, which are, intuitively, the maximal subsets of the relations in which navigation "makes sense." The maximal object idea is discussed in [MAIE81] and [ULLM82a, b], while the notion of paths that "make sense," which we take to mean "have a lossless join," is discussed in [AHO79c].

with a more intuitive response. In particular, the system does not eliminate a response just because certain tuples fail to appear in the join because their values do not match values in the relations that are extraneous to the query.

Example 4: Let us consider a database with attributes E, S, D, and M, standing for employee, salary, department, and manager, respectively, organized into three relations, ES, ED, and DM. To find the manager of Jones, we could write in System/U

retrieve (M) **where** E = 'Jones'

The System/U language is similar to QUEL (see [ULLM82b]), but since all tuple variables range over the universal relation, there is no need to declare them with range statements. Moreover, since many queries, such as the above, require only one tuple variable, we use an attribute A by itself to stand for blank.A, where "blank" stands for the default, or *blank tuple variable*.

In Fig. 3 we see the tableau representing the query in which the three relations ES, ED, and DM are joined, and the selection implied by the where-clause and the projection implied by the retrieve-clause are then applied. Notice how we take the natural join of relations by choosing one symbol for each attribute (e for attribute E, and so on). For each relation we create a row that has these symbols in the columns for those attributes the relation has and blanks elsewhere. Also, the selection clause is reflected by a condition appended to the tableau; some conditions of the form $A = B$ can be reflected by equating the symbols corresponding to attributes A and B. Finally note how the projection of the retrieve-clause is represented by the fact that only the symbols for the retrieved attributes appear in the summary.

	E	S	D	M
				m
(1)	e	s		
(2)	e		d	
(3)			d	m

e = 'Jones'

Fig. 3. Tableau for sample query.

We can reduce the tableau of Fig. 3 by mapping rows (1) and (2) to (2), and (3) to itself. In so doing, we map symbol s to the symbol represented by the blank in row (2), column S, which is legal. Note that we could not, for example, map row (3) to row (2) because m, since it appears in the summary, cannot be mapped to any other symbol. Thus, the minimum tableau weakly equivalent to Fig. 3 is the one shown in Fig. 4.

E	S	D	M
			m
e		d	
		d	m

e = 'Jones'

Fig. 4. Mininum-row tableau.

The tableau of Fig. 4 comes from the algabraic expression $\pi_M(\sigma_{E='\text{Jones}'}(ED \bowtie DM))$.† In comparison, the original tableau of Fig. 3 came from a similar expression that had the relation ES included in the join. The two expressions are weakly equivalent, as they must be, but they happen not to be strongly equivalent. We claim that the expression of Fig. 4 should be preferred to that of Fig. 3 for the following reason. The differences occur when Jones has a department listed for him in the ED relation, and that department has a manager listed in the DM relation, but we have no salary listed for Jones in the ES relation. Fig. 3 will not produce a manager for Jones, because the join $ES \bowtie ED \bowtie DM$ has no tuples for employee Jones. On the other hand, Fig. 4 will produce the manager or managers of the department or departments Jones works in.

† See [ULLM82b] for details about how we translate between tableau notation and ordinary relational algebra.

We claim that the latter interpretation is with high probability the one that the user intended. It is very unlikely that the interpretation of Fig. 3, which is "print the manager(s) of the department(s) of Jones unless we don't have a salary for Jones, in which case print nothing," is what the user intended. □

The reader should note that minimization under weak equivalence is essential for the transformation from Fig. 3 to Fig. 4 to take place. Without using weak equivalence, we could not produce plausible interpretations of many queries over universal relations, for the reason illustrated in Example 4: missing tuples in relations that are intuitively outside the path connecting the attributes mentioned in the query (as ES does not serve to connect E with M) would cause information to disappear from the answer. It is also worth noting that for this reason we cannot simulate a universal relation system by an ordinary system with a view facility. Even if we defined the universal relation to be a view consisting of the join of all the relations, we could not eliminate these extraneous terms from the join in an ordinary optimization phase, because such optimization would be performed according to the usual (strong equivalence) reduction rules.

Example 5: Let us briefly give another example of how System/U handles queries; this time we have a situation where more than one tuple variable is needed, the old query about the employees that earn more than their managers. In System/U this query is

retrieve (E) **where** $M = t.E$ **and** $S > t.S$

Here, we use attributes by themselves to correspond to attributes of the blank tuple variable, and we use another tuple variable t, as well. Think of the blank tuple variable as representing the tuple in the universal relation over $ESDM$ that corresponds to the employee, while t represents the tuple of the manager, as an employee. The clause $M = t.E$ ensures that the employee attribute for tuple t will be the same as the manager attribute in the tuple corresponding to the blank tuple variable.

Figure 5 shows the tableau that represents this query. Subscript 1 is used for attributes corresponding to the blank tuple variable, and subscript 2 is used for attributes of t. The constraint $M = t.M$ is represented by using the same symbol in the M_1 and E_2 columns, while constraint $S > t.S$ becomes $s_1 > s_2$ and is appended to the tableau. We reduce the tableau by sending rows (5) and (6) to (4), and all other rows to themselves. Note that we must send (5) and (6) to the same row, or else d_2 would be mapped to two different symbols. Also, we cannot reduce this tableau further. For example, if we tried to map (4) to (1), (2), or (3), the constraint $s_1 > s_2$ would not be implied by conditions on the symbols we mapped s_1 and s_2 to. In particular, s_2 would be mapped to the symbol represented by one of the blanks in the S_2 column, and there are no constraints involving any such symbols.

	E_1	S_1	D_1	M_1	E_2	S_2	M_2	D_2
	e_1							
(1)	e_1	s_1						
(2)	e_1		d_1					
(3)			d_1	m_1				
(4)					m_1	s_2		
(5)					m_1		d_2	
(6)							d_2	m_2

$s_1 > s_2$

Fig. 5. Tableau asking for employees who make more than their managers.

The expression represented by the reduced tableau having only rows (1)–(4) of Fig. 5 is computed by the following steps.

1. Take the natural join of ES, ED, and DM.
2. Take the equijoin of the result of (1) with ES, with M of the first equal to E of the latter.
3. Select for the condition that the first S component is greater than the second S component.
4. Project the result onto the first component (E).

Of course, this query must still be subjected to ordinary optimization (using the strong equivalence criterion) if it is to be implemented in an efficient way. However, the first stage, where we did tableau reduction under weak equivalence, was essential for us to get the right expression in the first place. □

V. Summary of the System/U Query Interpretation Algorithm

To summarize the roles of optimization under weak and strong equivalence in a universal relation system, let us list the steps of the System/U algorithm. The description we give is a simplification of the actual algorithm, since here we talk about relations, rather than "objects," which are the fundamental relationships among attributes, regardless of whether or not they correspond to relations. Also, "maximal objects" are, for our purposes, collections of relations in which we can find unique shortest paths connecting attributes, as, for example, the path from E to D to M is the shortest way to connect E and M in Example 4. For more details, see [ULLM82a, b].

1. For each tuple variable mentioned in the query, including the blank, assign a copy of the universal relation. Begin by writing down the expression that is the Cartesian product of all these universal relations (as we did in Example 5).
2. Modify the expression of (1) by applying to it the selection and projection implied by the retrieve- and where-clauses of the query.
3. Substitute for the copy of the universal relation associated with tuple variable t the union of all the maximal objects whose attributes include all those attributes A for which $t.A$ (or just A, if t is "blank") appears in the query.
4. Substitute for each maximal object the natural join of all the relations in that maximal object.
5. Distribute the Cartesian products, selections and projections over the unions, so we have a union of terms each of which has a tableau.
6. Using the weak equivalence criterion, reduce each tableau in this union. The method of [KUPE82] is used.
7. Also under weak equivalence, use the technique of [SAGI80] to eliminate redundant terms of the union.
8. Apply optimization under strong equivalence to the resulting expression.

References

[AHO79a] Aho, A. V., Y. Sagiv, and J. D. Ullman, "Equivalence of relational expressions," *SIAM J. Computing* **8**:2 (1979), pp. 218–246.

[AHO79b] Aho, A. V., Y. Sagiv, and J. D. Ullman, "Efficient optimization of a class of relational expressions," *ACM Transactions on Database Systems* **4**:4 (1979), pp. 435–454.

[AHO79c] Aho, A. V., C. Beeri, and J. D. Ullman, "The theory of joins in relational databases," *ACM Transactions on Database Systems* **4**:3 (1979), pp. 297–314.

[CODD70] Codd, E. F., "A relational model for large shared data banks," *Comm. ACM* **13**:6 (1970), pp. 377–387.

[CHAN77] Chandra, A. K. and P. M. Merlin, "Optimal implementation of conjunctive queries in relational databases," *Proc. Ninth Annual ACM Symposium on the Theory of Computing*, pp. 77–90.

[KLUG81] Klug, A., "Inequality tableaux," Dept. of C. S., Univ. of Wisconsin, to appear in *JACM*.

[KUCK82] Kuck, S. M. and Y. Sagiv, "A universal relation database system implemented via the network model," *Proc. ACM Symposium on Principles of Database Systems*, pp. 147–157.

[KUPE82] Kuper, G., "An algorithm for reducing acyclic hypergraphs," STAN-CS-82-892, Dept. of C. S., Stanford Univ.

[MAIE81] Maier, D. and J. D. Ullman, "Maximal objects and the semantics of universal relation databases," STAN-CS-81-878, Dept. of C. S., Stanford Univ.

[SAGI80] Sagiv, Y. and M. Yannakakis, "Equivalences among relational expressions with the union and difference operators," *J. ACM* **27**:4 (1980), pp. 633–655.

[SAGI81] Sagiv, Y., "Can we use the universal instance assumption without using nulls?," *ACM SIGMOD International Symposium on Management of Data*, pp. 108–120, 1981.

[SAGI82] Sagiv, Y., "A characterization of globally consistent databases and their correct access paths," unpublished memorandum, Univ. of Illinois, Dept. of C. S.

[ULLM82a] Ullman, J. D., "The U. R. strikes back," *Proc. ACM Symposium on Principles of Database Systems*, pp. 10–22.

[ULLM82b] Ullman, J. D., *Principles of Database Systems*, second edition, Computer Science Press, Potomac, Md.

Query Optimization Research in the

Database Programming Languages (DBPL) Project

+ * * *

M. Jarke, J. Koch, M. Mall, J.W. Schmidt

* Universitaet Hamburg
Fachbereich Informatik
Schlueterstr. 70
D-2000 Hamburg 13
Federal Republic of Germany

+ New York University
GBA - CRIS
90 Trinity Place
New York, N.Y. 10006
U S A

1. The DBPL Project

The DBPL project at the University of Hamburg is centered around the integration of database models and programming languages. The relational approach to databases and Pascal-like programming languages, both known for their well-designed data structuring capabilities, have proven as a framework particularly suitable for that integration effort.

The DBPL project evolved from ideas resulting from the design and implementation of the database programming language Pascal/R [SCHM77], [SCHM80] from 1975 to 1979. The DBPL project has the objective to investigate in more depth the issues of query optimization and concurrency control in database programming languages. The main goal is the design, implementation, and evaluation of language constructs that support shared access to databases and query optimization for calculus-oriented languages.

The DBPL approach is different from other approaches to query language design. Whereas most of the common interactive query languages are directed towards the novice user [VASS82], database programming languages address the professional programmer who needs a powerful tool for sophisticated database access and manipulation. We believe that the homogeneous extension of a programming language by appropriate data types and operations [SCHM77],[SCHM78] is a better way to support such a programmer than providing subroutine calls or heterogeneous imbedding of a query language in a programming language.

The usefulness of our approach is exemplified by several successful applications of Pascal/R. The system is used for teaching database management courses, for database applications (50 Mbytes) in fishery research [BIOM81], for the development of database programming methodologies [BROD81], and as a target language for very high level languages like TAXIS at the University of Toronto, and the natural language system HAM-ANS at the University of Hamburg.

The DBPL project (principal investigator: Joachim W. Schmidt)
is supported by the Deutsche Forschungsgemeinschaft (DFG)
under grant no. Schm. 450/2-1.

2. Query Optimization in Pascal/R

A Pascal/R programmer can query a database using a relation-valued expression with a selection predicate. The predicate is a well-formed formula of an applied many-sorted first-order calculus with existential and universal quantifiers where the "sorts" are the range relations to which a tuple variable is bound.

In one-sorted predicate calculus, quantifiers can be moved over terms in which the quantifier does not occur. This is used in all algorithms for standardization and optimization of predicates. In the many-sorted calculus, there are two cases where the result of such a transformation depends on whether the range relation of the quantified variable is empty [JARK81],[JARK82]. Therefore, the runtime system must be able to change the standardized and optimized query before execution.

Many approaches to query optimization first translate a calculus expression into a sequence of algebra operations and then optimize this sequence. In contrast, our strategies are described by transformations of relational calculus expressions and only the transformed expression is translated into operations [JARK78],[JARK82],[KOCH79]. This high-level approach allows for an extension and different interpretation of query optimization algorithms. First, as we do not predetermine the evaluation of subexpressions, parallel or concurrent evaluation is supported [SCHM79]. Second, we introduce the concept of extended range expressions which allows a tuple variable to be bound to subrelations rather than whole database relations [JARK82]. The application of this concept leads to predicates which can be evaluated more efficiently by avoiding repetitive evaluation of identical subexpressions in the case of existentially quantified variables and by reducing the number of conjunctions in the case of universially quantified variables. Finally, the relational calculus interpretation of tree queries leads to an extension of well-known semijoin algorithms to the cases of universal quantifiers and inequality comparison operators [JARK82].

To support this high-level approach, we introduced the language construct "reference" [JARK81] which can be used to define intermediate results like indexes or links. The advantage of having such a high-level tool for database system programming is that it allows the complete query evaluation process to be described as a nested relational calculus expression rather than as a sequence of operations.

Two extensions to the Pascal/R query optimization approach are still being investigated. First, we want to simplify predicates prior to their evaluation by analysing tautologies, contradictions, and idempotency. Second, the concept of extended range expressions will be generalized and used for access path selection.

3. Query Optimization Activities in the DBPL Project

The query optimization strategies in Pascal/R consider only one query at a time. Presently, we are working on extending the scope of query optimization from one statement to a set of statements. In database programming languages where concurrent access to databases is supported by tools for the formulation of compound database operations known as transactions, there are two lines of attack. First, we are working on the simultaneous optimization of all queries contained in a transaction, and second, we are investigating the idea of a shared query optimizer that processes queries of all active transactions in parallel.

Another activity is the exploration of advanced access methods with respect to their impact on query processing algorithms. We are especially interested in multi-dimensional access methods, i.e., access methods that efficiently support access over attribute combinations, range queries and (possibly) partially specified queries. We expect a substantial simplification of query processing algorithms from the use of those methods since the low-level tuple interface to the underlying access method can then be replaced by a high-level set interface.

Finally, the design of a language construct called "selector" [MALL82] for general access to selected parts of relations plays a central role in the DBPL project. The definition of a selector introduces a selector name and parameters, binds the selector to a relation, and provides a selection predicate. Selectors can serve as a relation-like description of access path to that part of a relation which fulfills the selection predicate. Selectors provide high-level tools for access path definition, maintenance, and use that fit particularly well in a relational query optimization system.

4. Current status of the DBPL Project

Query optimization and concurrency control algorithms are being implemented on a VAX-11 under VMS. The implementation tool is the system programming language Modula-2 which our group moved from a PDP-11 to the VAX-11 and adapted to the VMS environment [KOCH82].

Project participants in addition to the authors of this paper are Winfried Lamersdorf, Peter Putfarken, and Manuel Reimer.

References

[BIOM81] "Post-FIBEX Data Interpretation Workshop", BIOMASS Report Series No. 20, SCAR/SCOR/IABO/ACMR, Hamburg, September 21st to October 9th, 1982.

[BROD82] Brodie, M.L. "On Modelling Behavioural Semantics of Databases", Proc. 7th VLDB Conf., Cannes, 1981.

[JARK82] Jarke, M., Schmidt, J.W. "Query Processing Strategies in the Pascal/R Relational Database Management System", Proc. ACM/SIGMOD International Conference on Management of Data, June 2-4, 1982, Orlando, Florida, USA.

[JARK81] Jarke, M., Schmidt, J.W. "Evaluation of First-Order Relational Expressions", Universitaet Hamburg, Fachbereich Informatik, Bericht Nr. 78, June 1981.

[JARK78] Jarke, M. "Design and Implementation of Query Algorithms for Relational Database Systems" (in German), Fachbereich Informatik, Universitaet Hamburg, diploma thesis, 1978.

[KOCH82] Koch, J., Mall, M., Putfarken, P. "Modula-2 for the VAX: Description of a System Transfer" (in German), Proc. of the Meeting of the German Chapter of the ACM on Pascal-like Programming Languages, Kiel, July 1982.

[KOCH79] Koch, J. "On the Formulation and Integration of Predicates and Relations in Pascal/R" (in German), Fachbereich Informatik, Universitaet Hamburg, diploma thesis, 1979.

[MALL82] Mall, M., Reimer, M., Schmidt, J.W. " Data Selection, Sharing, and Access Control in a Relational Scenario", Symposium on Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages, Intervale, New Hampshire, June 17-20, 1982, to be printed in "Perspectives on Conceptual Modelling", Springer-Verlag, 1982.

[SCHM80] Schmidt, J.W. und Mall, M. "Pascal/R Report", Fachbereich Informatik, Universitaet Hamburg, Bericht Nr.66, January 1980.

[SCHM79] Schmidt, J.W. "Parallel Processing of Relations: A Single-Assignment Approach", Proc 5th VLDB Conf., Rio de Janeiro, October 1979.

[SCHM78] Schmidt, J.W. "Type Concepts for Database Definition", in Shneiderman, B. "Databases: Improving Usability and Responsiveness", Academic Press, 1978.

[SCHM77] Schmidt, J.W. "Some High Level Language Constructs for Data of Type Relation", ACM Transactions on Database Systems, Vol.2, No.3, September 1977.

[VASS82] Vassiliou, Y., Jarke, M. "Query Languages - A Taxonomy", NYU Symposium on User Interfaces, New York, May 1982.

Distributed Query Compilation and Processing in R*

Dean Daniels*
Pui Ng*
IBM Research Laboratory
San Jose, California 95193

Introduction

R* is a prototype distributed relational database system being implemented for research purposes at the IBM San Jose Research Laboratory [WILL82]. Each site in an R* network runs an extended version of System R [ASTR76]. Objectives of the R* project include providing users with a single system image of the DDBMS for ease of use, and allowing autonomous control of participating sites for availability.

R* users access the distributed database using the SQL database language [CHAM76]. Just as System R made SQL programs independent of details of the physical storage of data, the single system image presented by R* will make SQL programs independent of the location of data. The multi-site atomic transactions provided by R* [LIND79] are another important aspect of the single system image presented to users.

Site autonomy means that local administrators and users can retain control of data stored at their own R* site [LIND80]. R* allows controlled and voluntary sharing of data between sites, but individual sites must be able to perform operations on local data even if they are not in communication with the rest of the distributed database. Thus, there can be no central services, such as deadlock detection or naming, in R*.

Query processing in R* is complicated by the single system image and site autonomy objectives, and by the non-procedural nature of SQL. A further complication is introduced because R* compiles SQL statements into low level programs. Compilation of database queries results in considerable performance improvements compared with interpretive execution [CHAM81].

Compilation in R*

Like System R, R* compiles SQL statements into low level programs called access modules which make calls to the storage system to execute queries. Query

*Authors' current addresses: Dean Daniels: Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213; Pui Ng: Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139

compilation verifies the rights of a user to access the requested data, selects a strategy for processing the query, and generates the access module to implement the strategy. In R* an access module is a distributed program, and parts of an access module are stored, together with a record of the objects (indexes and tables) on which they depend, in the databases at the sites which will execute those parts of the query.

Compilation is a complicated distributed operation in R* and is described in more detail in [DANI82B]. First, the site originating the query, called the master, parses the SQL statement, performs catalog lookup to obtain schema information needed for compilation, selects a global plan for query execution, and generates a low level access module for the execution of its portion of the query. The SQL statement and global plan are distributed to all other sites participating in the query's execution, which are called apprentices. Apprentice sites also perform parsing, catalog lookup, access planning and access generation. However, access planning at apprentice sites is guided by the global plan supplied by the master. The global plan contains enough information to coordinate the operations of all participants.

The distributed compilation strategy used by R* supports local autonomy by allowing each participant to check the SQL statement and independently generate an access module for its portions of the query's execution. Any site in the distributed database can be a global planner, so there is no reliance on centralized services. Because global access planning is performed by a single query participant, the query processing strategy selected is as efficient as the strategy a centralized compiler would produce.

Access Planning

The R* global access planner selects a query processing strategy which minimizes the total cost of executing a query. Query processing cost is estimated as a weighted sum of the CPU, disk I/O, and communications operations needed to execute the query. Consideration of local processing costs in addition to communications cost is one useful and somewhat novel aspect of the R* access planner.

To estimate the cost of a particular access strategy, the access planner applies statistics about the tables referenced by the query to a query processing cost model which is based on properties of the R* storage system and join methods [SELI80]. Using the model, the access planner selects an order of table accesses, a method for joining each table to the intermediate query result, and an access path (index) for each table.

When a table referenced by a query is remote, the access planner will consider two different methods for accessing it. The first method considered is to move the whole remote table to the desired site after performing local restrictions and projections. Alternatively, if the remote table participates in a join, then access requests containing join keys may be sent from the join site to the table's storage site and only records matching the join keys need be sent in reply. This latter method is a dynamic semijoin. The multi-site join methods used by R* are extensions of the merge join and nested loop join used in System R [SELI79]. For a join of two tables from different sites the access planner

considers performing the join at either table's storage site, or at some other site.

Recompilation

Compilation creates dependencies of the access module on internal database objects. The validity of an access module depends on the access paths it uses, the data objects it references, and the access privileges on the data objects. As a result, database actions may invalidate existing access modules and require recompilation [NG82]. In general, there are three types of database actions that may lead to invalidation of an access module, corresponding to the three types of dependencies.

In a distributed database environment, compilation may require the cooperation of many sites. Hence recompilation may also be a multi-site operation. However, there are situations in which recompilation can be done on a local basis (local recompilation) even if the original compilation involved more than one site. For example, if the access module is invalidated by dropping an access path at a particular site, recompilation can be done by re-generating the access module at the same site. Thus recompilation is kept local to the site where the invalidation originates. Local recompilation is less expensive than multi-site global recompilation and does not depend on the availability of other sites.

When the access module is invalidated by dropping an access path, local recompilation can always be used but on certain occasions it leads to substantial performance degradation of the compiled code. By examining the global plan generated in the original compilation and using heuristics, R* can identify those occasions and perform global recompilation. On the other hand, if the access module is invalidated by changes in data objects (for example, a table is migrated from one site to another), recompilation may require other sites to participate.

Current Status and Future Plans

The R* query compiler is currently being implemented. At present, the compiler is operational for a subset of the SQL language. The access planner considers all of the multi-site join methods mentioned above, and the access code generator implements some of these methods. Access module dependencies are recorded at each participant site, and global recompilation is currently used in all cases. Local recompilation is being implemented.

Future work on query compilation in R* will include the design and implementation of support for replicated and partitioned tables, and for protection views [DANI82A]. Once operational experience is gained, it may prove necessary to revise the access planner's cost model or to extend its repertoire of multi-site join methods.

Acknowledgements

R* is very much a group effort. In addition to the authors, current members of the R* project are: Greg Fischer, Laura Haas, Ruth Kistler, Bruce Lindsay, Guy Lohman, Yoshifumi Masunaga, C. Mohan, Patricia Selinger, Paul Wilms, and Robert Yost.

References

[ASTR76] M. M. Astrahan et. al., "System R: Relational Approach to Database Management," ACM Transactions on Database Systems, June 1976.

[CHAM76] D. D. Chamberlin et. al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," IBM Journal of Research and Development, November 1976.

[CHAM81] D. D. Chamberlin et. al., "Support for Repetitive Transactions and Ad-Hoc Queries in System R," ACM Transactions on Database Systems, March 1981.

[DANI82A] D. Daniels,. "Query Compilation in a Distributed Database System," Master's thesis, Department of EECS, MIT, Cambridge, MA, February, 1982. Also research report RJ 3423 IBM Research Laboratory, San Jose, CA.

[DANI82B] D. Daniels et. al., "An Introduction to Distributed Query Compilation in R*," Proceedings of Second International Symposium on Distributed Databases, Berlin, September, 1982.

[LIND79] B. G. Lindsay et. al., "Notes on Distributed Databases," Research Report, RJ 2571, IBM Research Laboratory, San Jose, CA., July 1979.

[LIND80] B. G. Lindsay and P. G. Selinger, "Site Autonomy Issues in R*," Research Report, RJ 2927, IBM Research Laboratory, San Jose, CA., September 1980.

[NG82] P. Ng, "Distributed Compilation and Recompilation of Database Queries," Master's thesis, Department of EECS, MIT, Cambridge, MA, January, 1982. Also research report RJ 3375 IBM Research Laboratory, San Jose, CA.

[SELI79] P. G. Selinger et. al., "Access Path Selection in a Relational Database Management System," Proceedings of ACM-SIGMOD 1979.

[SELI80] P. G. Selinger and M. E. Adiba, "Access Path Selection in Distributed Database Management Systems," Proceedings of the International Conference on Databases, Aberdeen, July 1980.

[WILL82] R. Williams et. al. "R*: An Overview of the Architecture." Proceedings of Second International Conference on Databases, Jerusalem, June, 1982. Also research report RJ 3325, IBM Research Laboratory, San Jose, CA., December 1981.

A METHODOLOGY FOR QUERY OPTIMIZATION IN DISTRIBUTED DATABASE SYSTEMS

Chin-Wan Chung
Keki B. Irani

Program in Computer, Information and Control Engineering
and Department of Electrical and Computer Engineering
The University of Michigan
Ann Arbor, Michigan 48109
Phone: 313-764-8517

ABSTRACT

In this paper, we outline the current research on query optimization in distributed database systems at the University of Michigan. Specifically, we describe by example our model for representing sets of values of attributes generated while processing a query by a sequence of semijoin operations. This model provides an efficient methodology for deriving the estimates of the cardinalities of these sets which are needed to compute the cost of query processing. Further, we mention the intuitive ideas behind our heuristics for deriving a sequence of semijoins. Comparative results with the existing algorithms are provided.

1. INTRODUCTION

The concept of distributed database systems has emerged as a natural solution to the information processing problems of geographically dispersed organizations. We consider a distributed relational database system on a point-to-point packet switching communication network. The data transmission delay in such a communication network is roughly proportional to the quantity of data transmitted.

In order to process a query which needs to reference data from multiple sites, portions of the database at other sites have to be transferred to the user's site. Since the data transfer rate between sites in communication networks can be slow [WONG77], the minimization of the inter-site data transfer can be of importance in processing a distributed query.

The usual methodology for distributed query processing (DQP) consists of reducing the referenced relations using a sequence of semijoins (SSJ) after initial local processing. The semijoin strategy involves the following tasks:

(1) Estimation of the size of the relation reduced by each semijoin of a SSJ. Since it is usually assumed that the reduction of a relation is proportional to the reduction of the set of values of its attribute, this task is that of estimating the reduction of the latter set.

(2) Design of an algorithm to determine an optimal SSJ which incurs minimal total inter-site data transfer.

We discuss these problems in the subsequent sections. For simplicity, we assume that a relation is a unit of distribution and consider conjunctive equi-join queries.

The earliest work in the area of DQP is by Wong [WONG77]. The semijoin strategy has been suggested in the literature [BERN81, CHIU80, CCA80, HEVN79a, HEVN79b]. Under the assumption that the attributes are independent [CHIU80, CCA80, HEVN79a, HEVN79b], the reduction of relations by an arbitrary SSJ cannot be estimated accurately. An improved estimation method was introduced in [BERN81]. In Section 2, we describe our own model deriving these estimates. We think our model is simpler to implement. As far as the DQP algorithm is concerned, the one suggested in [BERN81] is based on the hill-climbing technique with two enhancements to the basic algorithm. In Section 3, we outline our algorithm and give some comparative results.

2. ESTIMATION OF THE CARDINALITY OF A REDUCED RELATION

To explain our model by an example, let us consider four relations: SUPPLIER (S#, S_NAME), CITY (C_NAME, S#), SUPPLY (S#, P#), and PART (P#, P_NAME). Assume that after initial local processing, the remaining query is

```
FIND  (SUPPLIER.S_NAME, PART.P_NAME, CITY.C_NAME)
WHERE (SUPPLIER.S# = CITY.S#) AND (CITY.S# = SUPPLY.S#)
 AND  (SUPPLY.P# = PART.P#).
```

For convenience we represent the attributes SUPPLIER.S#, CITY.S# and SUPPLY.S# by a1, a2 and a3, respectively and SUPPLY.P# and PART.P# by b1 and b2, respectively. The equality relation partitions these attributes into blocks $\beta 1 = \{a1, a2, a3\}$ and $\beta 2 = \{b1, b2\}$. We represent the set of values immediately after initial local processing of the attribute ai by Ai for i = 1, 2, 3 and that of bi by Bi for i = 1, 2. The sets of values of attributes change as semijoin operations are performed. The cardinality $C(\alpha)$ of an attribute α, therefore, changes and is equal to the cardinality C(X) of the set X which represents its values at a particular instant. These sets of values of the attributes form the elements of lattices.

For our example, all the sets initially reachable by semijoins between attributes of the same block are represented by the two lattices shown in solid lines in the Figure 1. (The Di in the figure is the domain of the attributes in the block βi for i = 1, 2.) For example, the semijoin S1 from a3 to a1 reduces the set A1 to the set A1A3 which represents $A1 \cap A3$, and the cardinality C(a1) changes from C(A1) to C(A1A3). Again, the semijoin S2 from b2 to b1 reduces B1 to B1B2 and changes the cardinality C(b1) from C(B1) to C(B1B2). The semijoin S2, however, causes A3 to

reduce as well, because the attributes a3 and b1 are in the same relation. This causes the initial lattice of the block $\beta 1$ to be exten ded as illustrated by the dotted lines in the Figure 1. The new set formed is named A3A4 as it is also a subset of A3. The semijoin S2, therefore, also reduces the cardinality C(a3) from C(A3) to C(A3A4). As the semijoins are performed the lattices grow.

The estimations of the cardinalities can now be stated in terms of this model. If αi and αj are two attributes and if the sets of their values at any point in time are Xi and Xj respectively, then a semijoin from αj to αi reduces the cardinality of αi to a new value $Cn(\alpha i)$ which is given by

$$Cn(\alpha i) = C(g.l.b.(Xi, Xj))$$

We have shown that this value can be calculated from the old value $C(\alpha i)$ by the following formula:

$$C(\alpha i) \times C(\alpha j) = Cn(\alpha i) \times C(l.u.b.(Xi, Xj))$$

A recursive algorithm is written to compute C(l.u.b.(Xi, Xj)).

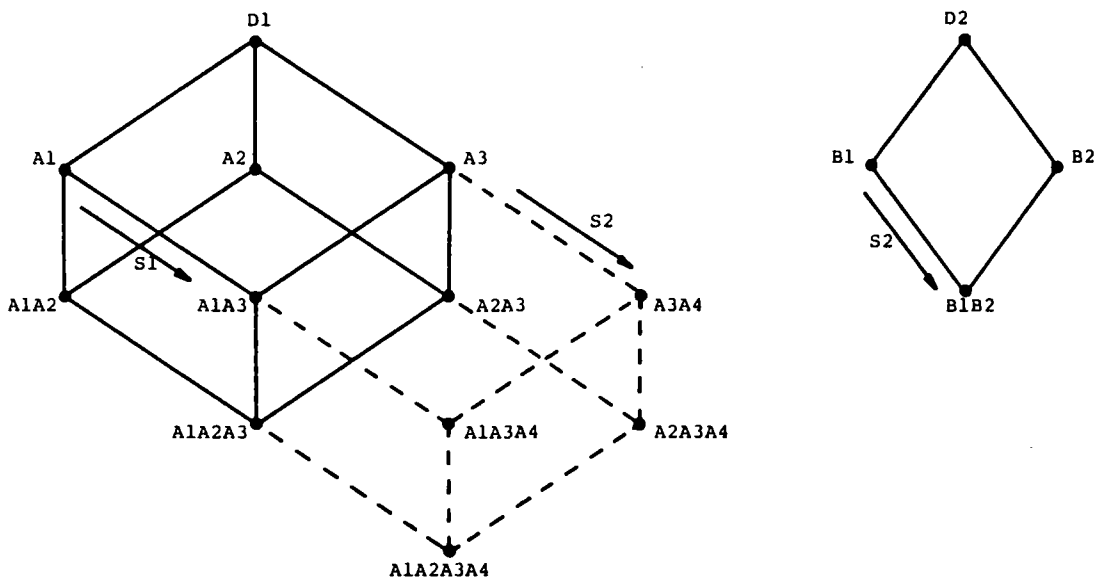


Figure 1

If αh is an attribute in the same relation as αi, its cardinality will also be reduced. Suppose $C(\alpha h) = m$. If n and k are the cardinalities of the relation before and after the semijoin, then the new cardinality $Cn(\alpha h)$ is given by

$$\begin{cases} m \times (1 - (1 - k/n)^{n/m}) & \text{if } n/m < k \\ m \times (1 - (1 - 1/m)^{k}) & \text{otherwise} \end{cases}$$

This is our approximation to the formula given by [YAO77]. The error is less than 0.5% for a broad range of values of n, m and k. This is a closer approximation than the one suggested in [BERN81].

3. DISTRIBUTED QUERY PROCESSING ALGORITHM

The DQP problem is known to be NP-hard [HEVN79a]. Hence any realistic algorithm for determining a SSJ involves heuristics. Our algorithm is no exception.

The total cost of DQP consists of the cost of moving data to perform the semijoins and the cost of moving the reduced relations to the user's site for the final processing. The direct benefit accrued by a semijoin is the difference between the amount by which it reduces a relation and the amount of data transported for its execution. A semijoin may not be directly beneficial at all or it may not have significant direct benefit. However, it may cause substantial benefits for the subsequent semijoins. That is why the straight-forward hill-climbing technique does not produce a good SSJ. Our algorithm uses a heuristic cost function which helps search for such semijoins which can generate indirect benefits.

The main feature of this algorithm is to select and "visit" an unvisited block with the least value of a heuristic cost function defined on the set of blocks until no more unvisited block exists. By "visiting" a block we mean that semijoins are scheduled which go from the attribute with the smallest cardinality in the block towards the one with the largest cardinality. After all the blocks have been visited, the visits are reversed. Several control features have been incorporated in the algorithm to increase the robustness for random input data.

For simple queries [HEVN79b], our algorithm produces optimal solutions. For the example in [BERN81], our algorithm and the algorithm in [BERN81] produce the same SSJ. For the example in [HEVN79b], the cost reported was 1324. We computed the cost of the same query using our algorithm and the algorithm in [BERN81]. The cost for the SSJ with our algorithm was 502 and the cost for the SSJ with the algorithm in [BERN81] was 796.

4. CONCLUSION

For a given initial database state, the cost of any sequence of semijoins to process a distributed query can be computed efficiently and effectively using the estimation method outlined here. A block-oriented heuristic algorithm has been developed to determine a low cost sequence of semijoins. The algorithms developed are being implemented in PASCAL.

REFERENCES

[BERN81] Bernstein, P.A. et al., "Query processing in a system for distributed databases (SDD-1)," ACM TODS, Vol. 6, No. 4, Dec. 1981, pp. 602-625.

[CHIU80] Chiu, D.M. and Ho, Y.C., "A methodology for interpreting tree queries into optimal semi-join expressions," Proc. 1980 ACM SIGMOD Conference, May 1980, pp. 169-178.

[CHUN82] CHUNG, C.W., "Query optimization in distributed database systems," Ph.D. dissertation, The University of Michigan, 1982.

[CCA80] Computer Corporation of America, "A distributed database management system for command and control applications: Final technical report - Part II," Technical Report No. CCA-80-04, Jan. 1980.

[HEVN79a] Hevner, A.R., "The optimization of query processing on distributed database systems," Ph.D. dissertation, Purdue University, 1979.

[HEVN79b] Hevner, A.R. and Yao, S.B., "Query processing in distributed database systems," IEEE Trans. on Software Eng., Vol. SE-5, No. 3, May 1979, pp. 177-187.

[IRAN82] Irani, K.B. and Khabbaz, N.G., "A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems," IEEE Trans. on Computers, Vol. C-31, No. 5, May 1982, pp. 419-434.

[WONG77] Wong, E., "Retrieving dispersed data from SDD-1: A system for distributed databases," Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, pp. 217-235.

[YAO77] Yao, S.B., "Approximating block accesses in database organizations," Comm. ACM. Vol. 20, No. 4, April 1977, pp. 260-261.

Distributed Query Processing
At Laboratoire IMAG

Nguyen Gia Toan

Laboratoire IMAG, B. P. 53x
38041 Grenoble, Cedex, France

1. Introduction

When a user query is submitted to a distributed database system, it usually has to be associated with some particular instances of the distributed data items (i.e., materializations), and broken into subqueries. These must be distributed and scheduled properly in order to produce the final result for the user. This paper sketches the initial transformations applied to user queries on a distributed database system (DDBMS) called MICROBE. MICROBE is operational at Laboratoire IMAG (University of Grenoble), on a local broadcasting network of LSI 11 machines. This network is dedicated to administrative and scientific applications.

Several techniques have been proposed for query processing, among which is the translation of high-level algorithmic languages into some QUEL-like internal form, on which the decomposition process is applied. This is the case in the SDD-1 project [GOOD79], where a query expressed in the Datalanguage is translated into QUEL statements, and further processed by a static query decomposition algorithm. Similarly, query management in the distributed version of the INGRES project dynamically decomposes QUEL statements into one-variable subqueries to handle the distributed execution of requests [STON80]. In both cases, however, the network of database computers is homogeneous. This allows high-level QUEL statements to be processed directly at the local sites.

A different approach has been implemented at Laboratoire IMAG for the POLYPHEME project [ADIB80]. The query, expressed in a relational algebra language, is translated into a binary tree of operators. This technique was proved very useful for the subsequent decomposition step, and particularly for the implementation of a dynamic decomposition strategy [NGUY82].

2. Query Optimization

The approach implemented for the MICROBE project combines the advantages of the above proposals. User statements are translated into binary trees of relational algebra operators and submitted to the optimizer. The optimizer restructures the query tree with respect to the properties of the relational algebra.

The query tree is restructured so that the data which is piped from each operator to the next one is minimized, and so that the total number of operations involved in the query is reduced. Restructuring is with respect to the relational algebra expression of the query, and does not take into account the distribution of relations. Techniques inherited from artificial intelligence are used for restructuring. They are related to the transformation of grammatically defined arborescences and to natural language recognition. The optimizer uses a set of catalogued transformation rules to

produce an equivalent query tree from its input node configuration.

MICROBE is the only DDBMS known so far to implement such a systematic optimization technique for user queries expressed in a high-level relational language. The advantage of this approach is that the optimizer can be driven by a set of simple transformation rules. These are applied to the query tree during several recursive scans. The principle of optimization is to isolate particular node configurations and apply the transformation rules where appropriate.

3. Query Decomposition

Once a query tree has been optimized, it is submitted to the decomposer. This module is in charge of the localization and fragmentation of the request into local subtrees. These are dedicated dynamically (during query execution) to particular execution locations at distributed database sites.

In previous proposals, the distributed execution strategy was entirely planned prior to the execution of any of its parts. In contrast, MICROBE relies on a dynamic decomposition strategy [NGUY81b], [NGUY82]. It does not require statistics on the database (except for cardinalities and widths of relations) or estimations of the size of the partial results produced by a query. (This kind of information is indeed difficult to maintain accurately, and costly to compute and retain, as shown in the usual static decomposition algorithms.) The system attempts to minimize a given cost function of CPU time and transfer costs of partial results [NGUY81b].

Dynamic query decomposition was first proposed for the POLYPHEME project. The designers of the distributed version of INGRES also implemented a dynamic query decomposition algorithm, which they proved to be usually more efficient than any static strategy.

The query decomposition process in MICROBE will be illustrated with an example. The tree resulting from the translation of a request is optimized via algebraic transformations. Suppose this yields the query tree of Fig. 1.

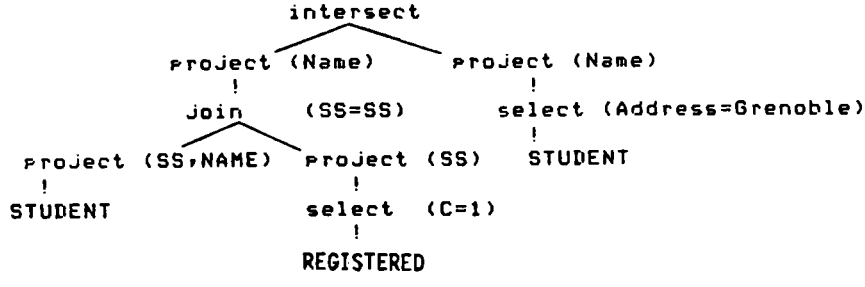


Fig. 1 Query tree to be decomposed.

3.1 Initial localization

The query tree is now ready for decomposition and distributed execution. The decomposition algorithm attempts first to localize as many nodes in the tree as possible. If the relations STUDENT and REGISTERED are located at sites S1 and S2 respectively, the lowermost monadic subtrees will be localized and sent for execution at S1 and S2 immediately (Fig. 2).

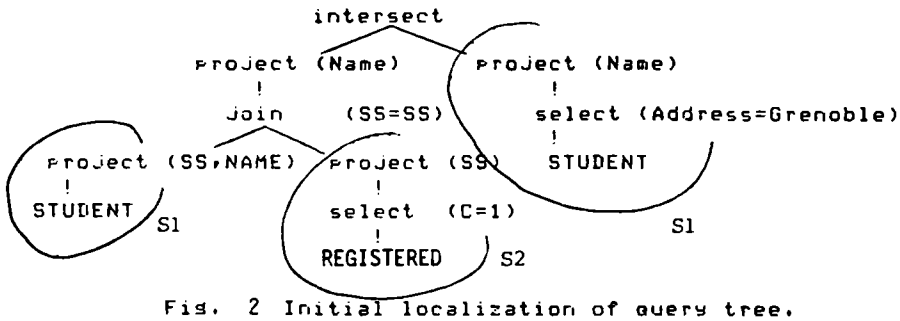


Fig. 2 Initial localization of query tree.

3.2 Dynamic localization

For the localization of the remining nodes (the INTERSECT, PROJECT, and JOIN operators), a dynamically updated threshold value is used to anticipate, whenever possible, the assignment of execution locations [NGUY81b]. This threshold is updated by a dynamic programming technique [NGUY82]. In the above example, if the volume of the result of the PROJECT (SS,NAME) on the STUDENT relation is less than the initial value T_0 of the threshold, the result will be transferred to the execution site of the brother subtree (PROJECT (SS) on SELECT (C=1) on REGISTERED), i.e. S2. The JOIN operator will therefore be assigned to site S2 for execution, as well as the PROJECT (NAME) operator (Fig. 3).

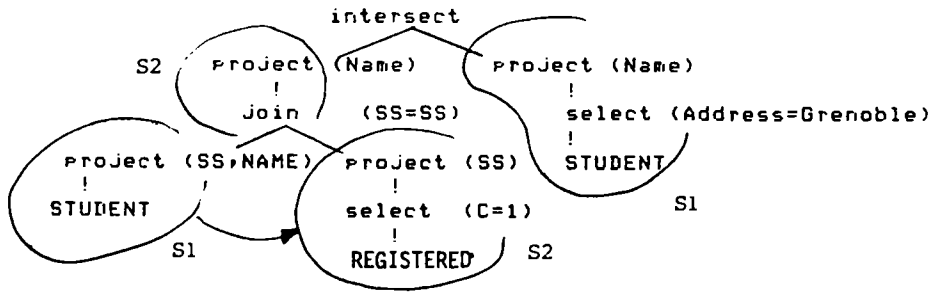


Fig. 3 Query tree after first decomposition step.

The initial threshold value T_0 is set to the average volume of the STUDENT and REGISTERED relations. It is then updated according to a predefined function. The new value T_1 and every localization decision are broadcast to all sites participating in the query, here S1 and S2.

The decomposition algorithm will subsequently wait for the first partial result R1 of the INTERSECT node operands to complete. It will then be possible to assign the execution location of the INTERSECT operator. If the volume of R1 is less than T_1, R1 will be transferred to the site of its brother subtree. Suppose R1 is the result of the PROJECT (NAME) on SELECT (ADDRESS=GRENOBLE). R1 will be transferred to S2. The INTERSECT operator is therefore scheduled at site S2 for execution (Fig. 4).

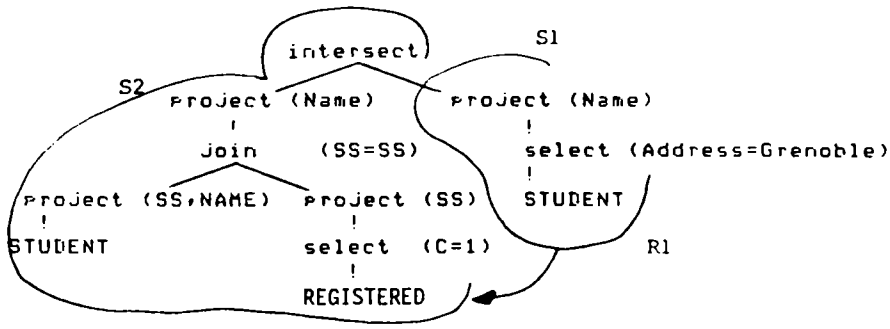


Fig. 4 Final decomposition of query tree.

3.3 Further comments

No hierarchical partitioning of database sites is done for query decomposition. All sites involved in the processing of a query, i.e. those sites where the relations are located, run the same decomposition algorithm. The initial query tree (Fig. 2) is first broadcast to all of them. Subsequent decomposition decisions are taken only when an exclusive privilege is held, implemented by a circulating token. This avoids conflicting decisions from two sits localizing simultaneously a common father operator [NGUY82].

It has been shown that this dynamic query decomposition algorithm is fully compatible with a dynamic data allocation strategy in a multi-user DDBMS [NGUY81b].

4. References

[ADIB80] M. Adiba et al, "An overview of the POLYPHEME distributed DBMS", Proc. IFIP 8th World Computer Congress, Melbourne, Oct. 1980.

[GOOD79] N.Goodman et al, "Query processing in SDD-1: a system for distributed databases", Computer Corp. of America Tech Report CCA 79-06, Oct. 1979.

[NGUY79] Nguyen, G.T., "A unified method for query decomposition and shared information updating in distributed systems", Proc. 1st International Conf. on Distributed Computing Systems, Huntsville, Oct. 1979.

[NGUY80] Nguyen, G.T., "Decentralized dynamic query decomposition for distributed database systems", Proc. ACM Pacific 80 Conf. San Francisco, Nov. 1980.

[NGUY81a] Nguyen, G.T., et al, "Distributed architecture and decentralized control for a local network database system", Proc. ACM International Computing Symp., London, March 1981.

[NGUY81b] Nguyen, G.T., "Distributed query management for a local network database system", Proc. 2nd International Conf. on Distributed Computing Systems, Paris, April 1981.

[NGUY81c] Nguyen, G.T., et al, "Local network databases", Computer Communications, Vol. 4, No. 4, August 1981.

[NGUY82] Nguyen, G.T., et al, "A high-level user interface for a local network database system", Proc. Infocom 82 Conf., Las Vegas, March 1982.

[STON80] M. Stonebraker, "Retrospection on a database system", ACM TODS, June 1980.

Research Directions in Query Optimization at the University of Maryland *

Alan R. Hevner
S. Bing Yao

Database Systems Research Group
College of Business and Management
University of Maryland
301 - 454-6258

1. Introduction

Improving the performance of data retrieval operations in file processing systems and database systems has remained an important goal since the advent of computers. As improved computer technology and new methods of viewing and structuring data have evolved over the years, researchers have developed algorithms for data access optimization based upon the new system environments. To make database systems more available to a wide range of users, relational query languages (e.g., SQL, QUEL, QBE) are being employed as a user-friendly interface for data access. Research today on query optimization is very active as evidenced by the number of papers on this topic presented at recent database research conferences such as SIGMOD and VLDB. Query optimization will remain a challenging research area in the future as new technologies, such as VLSI and networking, are used in database systems. New application environments for database systems, such as office information systems and engineering databases, will also provide new research impetus.

In this short paper, we overview our current research on query processing and optimization in the Database Systems Research Group at the University of Maryland. We discuss our recent results in the areas of query interfaces for database machines (Section 2), efficient join processing algorithms (Section 3), and query optimization in distributed systems (Section 4). In each section we provide a brief discussion of our future research directions.

2. Query Interfaces for Database Machines

Database machines have been proposed as hardware solutions to the problems of efficiency and reliability in many database systems [DATA 81]. The objective is to offload database management functions from the host computer to a specially designed, dedicated back-end computer whose sole function is to maintain the database and to process database requests. A database machine interfaces with the front-end host computer

* Portions of this research are supported by NSF Grant MCS81-07047.

through a high-level language interface. The major function of the host computer is to translate end user queries into the database machine language, send it to the database machine, receive results from the database machine and organize and display the results to the user. In principle we can deal with two separate, independent query languages in the system, the user query language and the database machine query language, with the host computer providing the intermediate translation step.

We have implemented a SQL-type interface [CHAM 76] for the IDM-500 relational database machine [BRIT 80]. Although both the query language and the database machine are based on the relational data model, discrepancies in their detailed operations made the implementation non-trivial. Based on this experience we presented a design for a database machine language interface in [LUO 82].

Database applications have many varieties. They differ in type of operations and data models. A database machine cannot possibly support all these access requirements directly. It is only feasible to provide a basic set of operations from which other operations can be derived. This set of operations must be complete, efficient, flexible, and extensible. They also must be non-procedural in order for the query access paths in the database machine to be used effectively. In view of these requirements, we have chosen to base the proposed database machine language on the relational data model. The advantage of the relational model is its simplicity and high level of operation.

The user interface language must be adapted to the application environment and the skill level of the user. We have investigated the problem of supporting user query languages based on the relational, hierarchical, and network data models. The translation from the user query language to the database machine language provides a level of implicit query optimization, since the database machine language is designed to make the most beneficial use of the machine's capabilities.

In particular, we have studied the office application environment. The users of an office information system are expected to be non-programmer professionals. The language that they use must be friendly enough for non-specialists to learn and use with a minimum of training. We have developed a screen oriented form query language [LUO 81]. The form data model [HOUS 76] allows a natural format for business and office information. In order to develop a simple user interface the idea of query-by-example [ZLOO 80] is applied to specify queries on forms. Additional procedural constructs are then introduced to enhance the capability of the language. The language is termed the Form Query Language (FQL). FQL can be shown to translate quite easily into a database machine language [LUO 82].

3. Database Join Processing

In relational query processing, clearly the most complex operation to optimize is the join. Since it is usually a time-consuming operation that accesses a large amount of data, finding efficient join processing strategies is an important method of improving database system

performance. In [YAO 79], a model of database query evaluation is presented from which the access costs of query processing can be analyzed and optimized. The optimization procedure decomposes an arbitrarily complex query into two-variable sub-queries. Methods for decomposition are usually heuristic in nature (e.g., [WONG 76]). An optimization procedure selects the least costly access strategy for each subquery by considering the detailed database storage structures and access costs. Join processing strategies are formed by different combinations of operation modules such as sorting, indexing, storage accessing, and merging.

A number of software methods for the processing of two-variable joins have been designed and implemented. Sort-merge techniques, nested loop techniques, together with the use of indexes provide a wide range of potentially beneficial join processing strategies [SELI 79, YAO 79]. These software approaches, however, have the limitations that large optimization programs must be implemented, controlled, and maintained.

Our recent research has developed the design of a two-dimensional join processor in hardware [TONG 81]. The join processor inputs values from the join attributes on different dimensions of a matrix structure. Comparators test equality at each matrix intersection and the join result is formed as output. We have compared this design with several other proposed join processor designs [TONG 82]. The proposed join processors can be classified into three categories: (1) one-dimensional array with pipelining, (2) one-dimensional array with broadcasting, and (3) two-dimensional array. Our analysis shows that the two-dimensional array approach has significant advantages in terms of both processing speed and hardware complexity. In addition, the two-dimensional array processor has a simple organization. The regularity of its design makes it suitable for VLSI implementation. A small-scale experimental VLSI chip has already been implemented. The experimentation of a more complete join processor array is being planned.

Our future research directions include extending cost models of two-variable joins to models that handle the analysis and optimization of n-variable joins where $n > 2$. The benefit of using the extended models for query optimization will be studied. The performance trade-off comes from the additional cost and complexity of optimizing an n-variable join versus the savings from reducing the number of decompositions required to break a query into n-variable sub-queries. As an extension we will also investigate the implementation of the n-variable join strategies in VLSI architecture.

4. Query Optimization in Distributed Systems

Distributed query optimization continues to be a major research effort in our group. In previous work we defined a cost model for query processing on distributed systems, developed optimization algorithms that we proved derive optimal processing strategies in certain system environments, and extended the optimization techniques into algorithms for general query environments [HEVN 78, HEVN 79a, HEVN 79b]. In [KERS 79] these optimization techniques are applied to a database allocated on a star network and the performance is analyzed. Our recent research has

extended this prior work in several ways.

We have developed an improved algorithm of polynomial complexity for the optimization of general queries [APER 82]. Three versions of the algorithm are presented; one for response time optimization and two for total time optimization. The response time version is proved to derive minimum response time processing strategies under the assumptions of our cost model. While neither total time version can guarantee to derive optimal total time strategies, an analysis shows that close-to-minimum total times are found for most queries.

Recent surveys of distributed query research have led to qualitative and quantitative analyses of proposed algorithms for query optimization. In [HEVN 82] a classification taxonomy is introduced. Algorithms are classified based upon the order in which the following optimization decisions are made.

a) Materialization - The selection of specific data copies to process the query.

b) Operation Order - The execution order of query operations as represented by a directed access graph.

In [SACC 81] algorithms are compared by comparing query performance on a common cost model. In both of these studies guidelines are given to identify the distributed system environments in which the different algorithms perform most effectively.

A future direction will be to extend our research to the optimization of database transactions. A transaction is a database application program which guarantees to maintain the consistency of a given consistent database state [GRAY 81]. Distributed optimization of a transaction differs from distributed query optimization. Requests for data retrieval are embedded within the procedural structures of a program, such as conditionals (IF-THEN-ELSE) and iteration (DO-WHILE). Particular emphasis in a transaction must be placed on points at which data is locked and unlocked and on commit points.

Our preliminary work on this question is reported in [HEVN 81]. None of the query optimization algorithms are directly applicable for transaction optimization because of the procedural constructs that limit the range of optimization in a transaction. For example, queries within a conditional statement may not be executed in a particular application run. Therefore optimizing their execution together with queries outside of the conditional may not be beneficial. Our proposed approach uses a two-step data flow analysis to form an effective transaction processing strategy. The goals of the optimization are to maximize parallel processing of data independent queries and to group data dependent queries into units of potential optimization within the transaction. A number of interesting research problems include the best use of parallelism on a network, the placement and storage of intermediate results during transaction execution, and the importance of recognizing common subexpressions in a transaction in order to minimize redundant processing in the system.

References

[APER 82] Apers, P., Hevner, A. and Yao S. "Optimization Algorithms for Distributed Queries," (to appear in IEEE Transactions on Software Engineering).

[BRIT 80] Britton-Lee Inc. "IDM 500 Intelligent Database Machine Manual," 1980.

[CHAM 76] Chamberlin, D. et al. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control," IBM Journal of Research and Development , Vol. 20, No. 6, Nov. 1976.

[DATA 81] Database Engineering "Special Issue on Database Machines," Vol. 4, No. 2, Dec. 1981.

[GRAY 81] Gray, J. "The Transaction Concept: Virtues and Limitations," Proceedings of the Seventh VLDB, Cannes, 1981.

[HEVN 78] Hevner, A. and Yao, S. "Query Processing on a Distributed Database," 1978 Berkeley Workshop, 1978.

[HEVN 79a] Hevner, A. and Yao, S. "Query Processing in Distributed Database Systems," IEEE Transactions on Software Engineering , Vol. SE-5, No. 3, May 1979.

[HEVN 79b] Hevner, A. The Optimization of Query Processing on Distributed Database Systems, Ph.D. Thesis, Department of Computer Sciences, Purdue University, December 1979.

[HEVN 81] Hevner, A. "Transaction Optimization on a Distributed Database System," Technical Report HR-81-259, Honeywell Corporate Computer Science Center, Bloomington, MN, May 1981.

[HEVN 82] Hevner, A. "Methods for Data Retrieval in Distributed Systems," Proceedings of the Second Symposium on Reliability in Distributed Software and Database Systems, Pittsburg, July 1982.

[HOUS 76] Housel, B. and Shu, N. "A High-Level Data Manipulation Language for Hierarchical Data Structures," IBM Research Report RJ 1756, March 1976.

[LUO 81] Luo, D. and Yao, S. "Form Operation By Example - A Language for Office Information Processing," Proceedings of the 1981 SIGMOD Conference, Ann Arbor, June 1981.

[LUO 82] Luo, D., Xia, D. and Yao, S. "Data Language Requirements of Database Machines," Proceedings of the 1982 NCC, Houston, 1982.

[SACC 81] Sacco, G. and Yao, S. "Query Optimization in Distributed Database Systems," Advances in Computers, Vol. 21, Academic Press, 1982.

[SELI 79] Selinger, P. et al. "Access Path Selection in a Relational Database Management System," Proceedings of the 1979 SIGMOD Conference, Boston, June 1979.

[TONG 81] Tong, F. and Yao, S. "Design of a Two-Dimensional Join Processor Array," Proceedings Sixth Workshop on Computer Architecture for Non-Numeric Processing, Hyeres, France, 1981.

[TONG 82] Tong, F. and Yao, S. "Performance Analysis of Database Join Processors," Proceedings of the 1982 NCC, Houston, 1982.

[WONG 76] Wong, E. and Youseffi, K. "Decomposition - A Strategy for Query Processing, " ACM Transactions on Database Systems , Vol. 1, No. 3, Sept. 1976.

[YAO 79] Yao, S. "Optimization of Query Evaluation Algorithms," ACM Transactions on Database Systems , Vol. 4, No. 2, June 1979.

[ZLOO 80] Zloof, M. "A Language for Office and Business Automation," IBM Research Report RC 8091, Jan. 1980.

Research on Query Optimization at Computer Corporation of America*

Umeshwar Dayal
Daniel Ries

Computer Corporation of America
575 Technology Square
Cambridge, MA 02139
(617) 491-3670

1. INTRODUCTION

In this paper we describe ongoing research on query optimization at Computer Corporation of America. This research is being carried out in the context of MULTIBASE and the two ADAPLEX database management systems. These are briefly described below.

The ADAPLEX LDM (Local Database Manager) and DDM (Distributed Database Manager) are systems that directly support a general-purpose database application programming language, ADAPLEX, which is the result of embedding the database sublanguage DAPLEX [SHIP81] in Ada.** A DDM query is mapped into a strategy consisting of single-site queries that are processed locally by the LDMs, and data movement commands for shipping the results of single-site queries between sites.

MULTIBASE is a system that provides a uniform, integrated query interface to a heterogeneous distributed collection of pre-existing databases. Database integration is accomplished by describing the schemas of the local databases in a common data model, DAPLEX, and then defining a global view tailored to the user's application over these DAPLEX representations; the view definition incorporates directives for resolving differences between the local databases [DAYA82a]. A user formulates queries in DAPLEX over his global view. A global DAPLEX query is first modified by the Global Data Manager (GDM) into a DAPLEX query over the local schemas. It is then decomposed into single-site DAPLEX queries that are shipped to the local sites. A Local Data Interface (LDI) at each site translates queries sent to it into queries (or programs) in the data language of the local host system. The GDM merges the results of these single-site queries into the final answer. The GDM has an ADAPLEX LDM available to it for managing schema and view definitions, storing auxiliary data required for database integration, and merging the results of single-site queries to produce the final answer.

---

*This research was jointly supported by the Defense Advanced Research Projects Agency of the Department of Defense (DARPA) and by the Naval Electronics System Command (NAVELEX) under contract N00039-82-C-0226. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing those of DARPA, NAVELEX, or the U.S. government.

**Ada is a trademark of the Department of Defense (Ada Joint Program Office).

The reader is referred to [SMIT81, LAND82] for overviews of MULTIBASE and to [CHAN81, CHAN82] for overviews of the ADAPLEX LDM and DDM.

2. QUERY PROCESSING IN THE LDM, DDM, AND MULTIBASE

One issue common to these three systems is the processing of DAPLEX queries, albeit in somewhat different environments. DAPLEX is a high-level procedural language. The processing of a DAPLEX query has two main steps: decompilation and optimization. Decompilation transforms the query into an envelope, which is a non-procedural internal representation of the data selection component of the query, and a postprocessing program that formats and prints the retrieved data. The principal advantage of decompilation is that it separately identifies the data selection requirements of the query, which can then be subjected to optimization. This allows for more optimization than is found in embedded relational query language systems such as PASCAL/R [SCHM77] and RIGEL [ROWE79].

Optimization of the envelopes is performed at two levels: local (in the ADAPLEX LDM and each MULTIBASE LDI) and distributed (in the ADAPLEX DDM and the MULTIBASE GDM). At the local level, the objective is to minimize local processing costs by proper access path selection. At the global level, the objective is to minimize the amount of data moved between sites and to exploit parallel processing at different sites. Local and distributed optimization are described in the next two sections.

3. LOCAL OPTIMIZATION

3.1 Optimization in the LDM

The LDM must support extensive optimization of single site ADAPLEX queries. The LDM optimizer incorporates several extensions to relational optimization techniques. Some of these extensions are required due to the richness of ADAPLEX and the functional data model. Other extensions could also be applied to relational languages embedded in programming languages.

The richness of ADAPLEX has two main implications for the LDM optimizer. First, unlike relational languages, ADAPLEX allows the user to explicitly control the grouping of entities and the elimination of duplicates. This grouping implies an order of performing joins. The LDM takes advantage of the explicit grouping specifications to maintain hierarchical temporaries. This use of hierarchical temporaries greatly reduces the size of data that must be sorted, saved on disk, and passed back to the user program. In addition, the LDM optimizer analyzes other join orders, and will, if cost effective, normalize the temporaries (into first normal form). Of course, the re-ordering and re-grouping that may be required after performing these alternative sequences of joins must be included in the associated cost estimates.

Second, the LDM optimizer takes advantage of the explicit entity-to-entity functions that can be specified in the functional data model. (For example, the "works-in" relationship may be represented by a function from the Employee entity type to the Department entity type.) These are implemented by pointers and serve as fast access paths for performing joins of the two entity types between which a function is defined. The LDM optimizer compares the costs of using and not using these fast access paths.

The LDM optimizer uses several additional optimization strategies that could be exploited by more ambitious optimizers for embedded relational queries. First, it attempts to optimize over several nested "for each" loops in the query. Many embedded relational query language systems such as PASCAL/R [SCHM77] and RIGEL [ROWE79] optimize the qualification of one "for each" loop at a time. Optimization over several loops allows the LDM to consider more join strategies, and can reduce the communication and synchronization overhead between a user program and the LDM. Second, the LDM uses a leveled hill climbing search strategy for limiting the number of join orders that it considers. The user can specify an optimization level, N; the optimizer enumerates join sequences of length N at each stage of optimization. When N is one, this strategy is very similar to the "greedy strategy" of INGRES [WONG76]. Finally, the LDM optimizer considers several strategies for processing arbitrarily nested universal and existential quantifiers.

For details of decompilation and optimization in the LDM, see [RIES82].

3.2 LDI Optimization in MULTIBASE

MULTIBASE must interface with a wide variety of Database Management Systems: some (e.g., CODASYL) have procedural, navigational data manipulation languages; others (e.g., ADAPLEX LDM and relational systems) have high-level query languages. Because these systems differ substantively in the level of direct control over access path selection that they give to programmers, different optimization techniques are necessary.

High-level query language systems are themselves equipped with query optimizers, and so MULTIBASE assumes that no optimization is necessary in the LDI; the LDI need merely transliterate the DAPLEX single-site query into a query in the local language.

Navigational systems, however, require a DML program to specify an appropriate route through the database. The LDI usually has a choice of programs, each of which specifies a sequence of access path traversals. In [DAYA82b] we describe an access path optimizer for navigational systems such as CODASYL. We have identified a class of queries, the tree queries, that can be effectively optimized. The LDI optimizer decomposes the single-site query sent to it into tree queries, enumerates the strategies for processing each tree query, estimates their costs, selects the cheapest strategy for each tree query, merges these into an overall strategy for the single-site query, and compiles this strategy into a loop program in the host DML. The analysis is considerably complicated by the presence of quantifiers in tree queries.

4. DISTRIBUTED OPTIMIZATION

4.1 Optimization in the DDM

The DDM optimizer extends distributed query optimization techniques used in other systems such as SDD-1 [BERN81] and System R* [SELI80]. Both SDD-1 and R* focus on conjunctive queries (i.e., queries involving only selections, projections, and joins). SDD-1's strategy is to reduce data by a sequence of semijoins, then ship the reduced data to a single site, where the result is produced by joining. R* and the DDM consider mixed join and semijoin strategies. The main difference is in the handling of horizontally partitioned data. Consider, for example, the query R $\bowtie$ S, where R is horizontally partioned into R1, R2; and S into

S1, S2. Both SDD-1 and R* replace the original query by the union of the four queries Ri ⋈ Sj, $1 \leq i,j \leq 2$. This is not always a good strategy. The DDM will consider the options of first performing one or both of the unions before the join (i.e., the possibility of left and/or right distributing the join over the union).

As with the LDM, additional extensions to the optimizer are necessitated by the richness of ADAPLEX: the presence of quantifiers, unidirectional outer joins, user control over duplicate elimination, etc.

The overall optimization technique is the following. First, localize selections. Then, enumerate join orders. For each join of horizontally partitioned entity sets, determine whether it is cost-effective to distribute the join over the union. For each non-local operation, select a site for performing the operation, and determine whether it is cost-effective to do semijoin reductions before each join. Work is currently in progress on developing a cost model, heuristics for strategy enumeration, and heuristics for selecting copies of replicated data to be used in processing a query. For details of DDM optimization, see [DAYA82c].

4.2 Global Optimization in MULTIBASE

The overall approach to global optimization in MULTIBASE is very similar to that in the DDM. However, there are some differences. First, instead of disjoint horizontal partitions, we may have overlapping data in two or more local databases. For example, two Employee databases may contain data on overlapping sets of employees. In fact, the overlapping data might be inconsistent. For instance, the salary values of an employee may be different in two databases. In the integrated global view, this inconsistency may be resolved by defining a generic entity type Employee, whose Salary value is defined to be some aggregate function (e.g., average or sum) of the Salary values in the two databases. Now we have to be careful in localizing selections or distributing joins over unions. For example, if the aggregate function used to define Salary is "average", then the selection of employees based on their salaries cannot be done completely locally at the two sites. We have derived a set of rules for distributing selections and joins over unions for various aggregate functions. Also, the reduction step considers the option of a "semiunion" reduction before performing a union. In our example, the semiunion partitions the employees in a local database into two subsets: those contained only in that database, and those having corresponding records in the other database. For the first subset, the selection can be performed locally; for the second subset, the salary values must be retrieved to perform the aggregation and selection in the GDM.

A second difference is that not all sites are powerful enough to process joins, unions, semijoins, semiunions, quantifiers, etc. Missing capabilities are compensated for in the GDM.

Finally, a global query may be posed over a global view that is defined over several intermediate levels of views, each involving generalization and aggregation. For these, query modification and global optimization proceed recursively from level to level. We depart from the usual approach to view processing, in which a query against the view is first modified all the way down to the local schemas, and only then optimized. To see why a different approach is necessary, consider a view, Employee, defined by generalizing two overlapping subtypes,

Employee1 and Employee2. In the usual approach, a query against Employee would be modified into the union of three subqueries against Employee1-Employee2, Employee2-Employee1, and Employee1 ∩ Employee2. However, if Employee1 and Employee2 are stored at separate sites, then it would be inefficient to separately optimize and execute these three subqueries. Our approach is to reduce (if possible and if cost-effective) Employee1 and Employee2 per the query's qualification and target list, then move them to the GDM, and partially materialize the view. If Employee1 is itself a generalization of overlapping subtypes stored at separate sites, this procedure is applied recursively to its subtypes.

For details of global optimization in MULTIBASE, see [DAYA82d].

5. REFERENCES

[BERN81] Bernstein, P.A., et al., "Query Processing in a System for Distributed Databases (SDD-1)," ACM TODS 6, 4 (Dec. 1981).

[CHAN81] Chan, A., et al., "The Design of an Ada Compatible Local Database Manager (LDM)," Technical Report CCA-81-09, Computer Corporation of America, Cambridge, Mass., Nov. 1981.

[CHAN82] Chan, A., et al., "The Design of an Ada Compatible Distributed Database Manager (DDM)," Technical Report, Computer Corporation of America, Cambridge, Mass., in preparation.

[DAYA82a] Dayal, U., and H.Y. Hwang, "View Definition and Generalization for Database Integration in MULTIBASE: A System for Heterogeneous Distributed Databases," Proc. Sixth Berkeley Workshop on Distd. DB Mgmt. and Comp. Networks, Feb. 1982.

[DAYA82b] Dayal, U., and N. Goodman, "Query Optimization in CODASYL Database Systems," Proc. ACM SIGMOD Intl. Conf. on Mgmt. of Data, June 1982.

[DAYA82c] Dayal, U., et al., "Query Optimization in the Distributed Database Manager (DDM)," Technical Report, Computer Corporation of America, Cambridge, Mass., in preparation.

[DAYA82d] Dayal, U., et al., "Global Query Optimization in MULTIBASE," Technical Report, Computer Corporation of America, Cambridge, Mass., in preparation.

[LAND82] Landers, T.A., and R.L. Rosenberg, "An Overview of MULTIBASE," Proc. Second Intl. Conf. on Distd. Databases, Berlin, Sept. 1982.

[RIES82] Ries, D., et al., "Decompilation, Optimization, and Pipelining for ADAPLEX: A Procedural Database Language," Technical Report, Computer Corporation of America, Cambridge, Mass., in preparation.

[ROWE79] Rowe, L.A., and K.A. Shoens, "Data Abstraction, Views, and Updates in RIGEL," Proc. ACM SIGMOD Intl. Conf. of Mgmt. of Data, May 1979.

[SCHM77] Schmidt, J.W., "Some High Level Constructs for Data of Type Relation," ACM TODS 2, 3 (Sept. 1977).

[SELI80] Selinger, P.G., and M. Adiba, "Access Path Selection in Distributed Database Management Systems," Proc. Intl. Conf. on Databases, Univ. Aberdeen, Aberdeen, Scotland, July 1980.

[SMIT81] Smith, J.M., et al., "MULTIBASE -- Integrating Heterogeneous Distributed Databases," Proc. NCC, Vol. 50 (Jan. 1981).

[WONG76] Wong, E., and K. Youssefi, "Decomposition -- A Strategy for Query Processing," ACM TODS 1, 3 (Sept. 1976).

Processing Multiple Queries in Database Systems[1]

Upen S. Chakravarthy and Jack Minker
Department of Computer Science
University of Maryland
College Park, Maryland 20742
Phone: (301)-454-4251

Abstract

Research activity on query evaluation and optimization has been centered around processing single queries. While a need for grouping queries for simultaneous evaluation on a database has been recognized, very little has been proposed. This paper discusses our ongoing research in that direction. The effect of grouping queries is to reduce local processing at a node by minimizing access to a particular relation. Similarly, data transfer across the nodes of a distributed database can be reduced. The result is enhanced system utilization. The process of decomposition and optimization of multiple queries is under investigation, as well as the characterization of environments where this grouping can be used effectively.

1. INTRODUCTION

The majority of present work on query processing and optimization pertains to processing of queries expressed in high level, non-procedural languages for the relational data model. See [KIM80] for references. Query processing systems, in general, attempt to minimize the cost of processing a set of queries by minimizing the cost of each query separately. Individual plans are generated for each query and executed on the database in succession. However, as shown in [GRAN80] it should be possible to group a set of queries over a database and minimize the cost of processing them as a unit. The commonality that exists among a set of queries, in terms of access to relations, join/semi-join operations and data transfers can be used to reduce the overall cost of their evaluation. Below we outline the general problem of multi-query evaluation and other associated problems and discuss results of preliminary investigations.

The main goal of query optimization has been to retrieve relevant data from a database with as little local processing and data transfer across nodes as possible. The criteria for optimization has varied widely in the literature, ranging from minimizing CPU time in centralized databases to parallel transfer of information and queueing delays in distributed systems. Also database semantics have been used to reduce queries to semantically equivalent queries, which can be evaluated more efficiently. See [KING81] for a detailed discussion and additional references.

2. PROBLEM CHARACTERIZATION

Grouping a set of queries for simultaneous evaluation allows common intermediate results to be computed and/or transferred only once. Joins and semi-joins occurring in several queries can be carried out simultaneously, accessing relations in common only once. This could substantially reduce the local processing required. Common intermediate data for different queries can be grouped and transferred once, saving substantial data transfer time in distributed databases. Some environments where grouping several queries for simultaneous evaluation is meaningful and beneficial are:

- where transactions are submitted in a batch-processing environment;
- in an interactive environment where queries come in at a steady rate (queries within a small time interval can be grouped without having to delay the response);
- where a database supports several external views (a single query may turn out to be a disjunction of several queries on conceptual views, all of which can be processed simultaneously);
- in deductive databases, deductive axioms applied to a single query may give rise to several disjunctive queries, which can be grouped together.

Determining the utility of multiple query evaluation requires that several subproblems be investigated:

- determining the probability that the same relation name appears in a set of queries;

---

[1]This work was supported by NASA grant NAG-1-51 and by NSF grant MCS-7919418.

- estimating the optimum number of queries that need to be grouped for cost effective plan generation;
- developing decomposition and plan generation algorithms for a set of queries treated as a single unit;
- estimating intermediate result sizes for multiple query decomposition;
- characterizing processing environments where this kind of grouping is beneficial;
- estimating the cost difference in optimizing a set of queries sequentially and that of optimizing as a single unit;
- determining the overhead and bookkeeping involved in executing a single plan for a set of queries and distributing the answers to the rightful originators of the query;

Solutions to the sub-problems listed above will enable us to characterize the problem of multiple-query processing in general. For a detailed discussion refer to [CHAK82]. Development of good decomposition and plan generation algorithms including intermediate estimation is a central issue in this investigation. The effectiveness of the algorithm depends on the amount of common computation that can be performed on the grouped queries. The analysis of this algorithm and its comparison with sequential execution of queries does not seem to be straight forward. Intermediate bookkeeping required can be a deciding factor for the utility of this approach in general. If bookkeeping is prohibitive it may vitiate any advantage in processing efficiency achieved. Theoretical computation of the optimum size and the selection of queries for grouping does not appear to be straight forward. The optimum size computation is the difficult problem. However, in deductive databases, the disjuncts derived from a goal tend to have common computations and can be grouped without much analysis.

3. JUSTIFICATION FOR GROUPING QUERIES

In a preliminary investigation [CHAK82] we have considered a worst case situation in which queries are considered independent. In this event we are interested in determining whether there will be overlap of relation names for a set of given queries. This overlap in relation names can be converted into common retrieval operations and used to evaluate the set of queries efficiently. We have considered a simple model and computed the probability with which the same relation names appear in more than one independently generated queries against a relational model. We describe below a probabilistic model and the assumptions for computing the overlap probabilities.

Each query is assumed to be in the algebraic form

$$\pi_T \ \sigma_q \ (R1' \times R2' \times \ldots \times Rn'),$$

where π and σ are projection and selection operators respectively; T contains the attributes in the answer relation; q is a predicate (in the form of boolean expression of simple clauses); and each Ri' is either a relation in the database or derived from union or difference operations. The string (R1' X R2' X ... X Rn') is called the matrix of the query.

3.1 Definitions of Elements of the Probabilistic Model

Let DB be the database of interest.
Let N be the number of relations in the database DB.
Let qi denote the i-th query.

The physical length of the query q, denoted by Lq, is a positive integer count of the actual number of relation names present in the matrix of the query q. Relation names are counted as many times as they occur.

The commonality among a set of queries is expressed in terms of the "overlapping of relation names" and is defined as follows:

A relation name Ri is said to belong (occur) to (in) the query q, denoted by Ri ϵ q, if the relation name Ri appears in the matrix of the query q.

n queries q1, q2, ... , qn are said to one-overlap, if there exists at least one relation name Ri ϵ DB such that Ri ϵ qi for i = 1 to n.

Similarly, n queries q1, q2, ... , qn are said to k-overlap if there exists at least k relation names R1, R2, ... , Rk, each Ri ϵ DB for i = 1 to k and such that Ri ϵ qj for i = 1 to k and j = 1 to n.

n queries q1, q2, ... , qn are said to exactly k-overlap, if there exists

exactly k relation names R1, R2, ... , Rk, each Ri ε DB for i = 1 to k and such that Ri ε qj for i = 1 to k and j = 1 to n.

3.2 Assumptions about the Model

Assumptions about the elements of the model can vary according to the database usage environment. Possible assumptions are:

(a) Relationships among queries: Queries can be assumed to be statistically independent of each other. Another possibility is to assume that the queries are correlated in some specific manner.

(b) Length consideration of the queries: The physical length is used in all discussions in this paper. Other lengths such as the unique length where only distinct relation names are counted are also possible.

(c) Distribution of query lengths: The length distribution is an important factor in overlap estimation. Though the length of the query can be arbitrarily large theoretically, in practice queries tend to be small. The gamma distribution with parameters α and β has the following properties: There is a reasonable probability that a query with a small number of relations will occur, while for a query with a modest to large length the probability drops to zero rapidly. This seems to be a good first approximation for the query lengths. The values of α and β can be varied to approximate realistic situations. The gamma distribution is represented by

$$F(x) = \frac{e^{-\alpha x}\,\alpha(\alpha x)^{\beta-1}}{(\beta-1)!} \qquad \text{for } x \geq 0$$

for positive integer values of β. The mean and the standard deviation are β/α and $\sqrt{\beta}/\alpha$ respectively.

Based on the above assumptions the probability of k-overlap of n queries can be computed. However initially the probability of one-overlap has been computed for two queries under the assumption that the queries are independent and the gamma distribution approximates the physical length of the queries.

3.3 One-overlap Computation for Two Queries

Let the queries be q1 and q2. Let there be N relations in the database. The probability of one-overlap can be expressed by the following formula:

$$\text{Prob (one-overlap)} = 1 - \text{Prob (no-overlap)}$$

The Prob (no-overlap) can be expressed as:

$$\text{Prob (no-overlap)} = \sum_{L} \Big[\sum_{m} \text{Prob (no-overlap}|L1,L2,m1,m2) * \text{Prob (m1,m2}|L1,L2) \Big] * \text{Prob (L1,L2)}$$

under the constraint $1 \leq m1 \leq L1$ and $1 \leq m2 \leq L2$.

This formula expresses the no-overlap probability as the summation of the product of no-overlap probability for given lengths L1 and L2, for queries q1 and q2 respectively and the probability of mi distinct relation names appearing in qi. This is further summed over all possible length values to obtain the probability of no-overlap regardless of the lengths involved. m1 and m2 ensure that all queries of lengths L1 and L2 are separated into mutually exclusive classes so that their individual probabilities can be summed.

As noted previously, the gamma distribution is assumed to represent the length distribution. It is shown in [CHAK82] that the probability of no-overlap for a specific value of L1 and L2 is given by the formula:

$$\sum_{m} \text{Prob (no-overlap}|L1,L2,m1,m2) * \text{Prob (m1,m2}|L1,L2) =$$

$$\sum_{i=1}^{k} \frac{(N-i)^{L2}}{N^{L1+L2}} * \sum_{j=1}^{r_i} Kij * Mij$$

where k = min (N,L1)
ri = number of integer solutions of X1+X2+ ... Xi = L1; $1 \leq i \leq k$.

Kij = ways in which i distinct relation names can be chosen out of N relations for the j-th solution.
Mij = multinominal coefficient for the j-th solution with i distinct relations adding up to L1.

Figure 1. shows curves plotted with Prob (one-overlap) along the Y-axis and β (a parameter of the gamma distribution) along the X-axis for different values of N (the number of relations in the database) and α . This is only a representative graph which has been drawn for different values of α . The α and β values are significant in that they help adjust the distribution of the query lengths for a given database. The Figure shows, for example, a value of 0.4 for the probability of one-overlap for a database that contains ten relations, for the α value of 1 and β value of 4. This means that 40% of the time there will be at least one relation overlap among two randomly selected queries on the database. α and β values suggest the average lengths of queries for which this computation holds i. e., 4 in this case.

Let T1 be the average time taken to execute the plan for a single query and T2 be the average time taken to execute a single plan for two queries. The expected average time for evaluating 2 queries in these two cases are 2*T1 and (0.6*2*T1 + 0.4*T2) respectively. If T2 = p*T1, where 1 <= p <= 2 then the expected average time to evaluate two queries is 2*T1*(0.6 + 0.2*p). If p is substantially less than 2, there is an advantage. If p is as low as 1, then we have the expected average time for two queries is 0.8*(2*T1) or a savings of 20%. If p is 1.5 there is a savings of 10%. We have considered a worst case situation where queries are independent of one another.

4. DECOMPOSITION OF MULTIPLE QUERIES

A general algorithm for decomposition and plan generation of multiple queries is being developed. Queries which have overlapping relation names are grouped to start with and a single query graph (called the super-graph) is constructed. A super-graph or a multi-query graph can be thought of as the superimposition of individual query-graphs. From this graph it is possible to generate a plan exploiting the commonality among the queries. Below we illustrate the idea of a super-graph with an example. Given the following database:

BOOKS (Title, Author, Pname, Lc_no)
PUBL (Pname, Paddr, Pcity)
BORR (Name, Addr, City, Card_no)
LOANS (Card_no, Lc_no, Date) and queries

Q1: List the books that have been borrowed before 1/1/78, and
Q2: Find the borrower's name and publisher's name and city for all the books borrowed before 1/1/78.

The super-graph for Q1 and Q2 is as shown in Figure 2. This graph is used to generate a single plan for all the queries which make up the graph. The plan generation using this graph facilitates proper grouping of common retrieval operations.

5. SUMMARY

We have proposed an approach for reducing the cost of query evaluation on a given database by grouping queries. Techniques applied to optimizing individual queries, such as semantic information, syntactic information and knowledge based information can also be applied. We believe that grouping queries is promising in terms of the cost reduction possible in query processing.

Multiple query processing can be achieved at two levels. At the query level, several queries can be grouped to start with and a single plan generated. This paper provides a rationale for grouping queries as well as a formal justification. The discussion in section 3 indicates that even in the worst case where queries are independent of one another, there can be a gain in grouping queries for common retrieval. Secondly, parallel processing of several queries can be achieved in a limited manner at a lower level (interface to the physical database) by grouping plans that are to be executed at that point. This can help those cases where it is difficult to group queries at the query level. A preliminary algorithm has been developed to generate a plan for decomposing a set of queries.

We have described the problem of multiple query optimization in a general framework and discussed the results of preliminary investigation very briefly. More research is needed in terms of decomposition and plan generation algorithms, and the means to identify commonality among queries and group them. Estimation of

reduction in the cost of processing using this approach has to be compared with other optimization schemes to substantiate the approach.

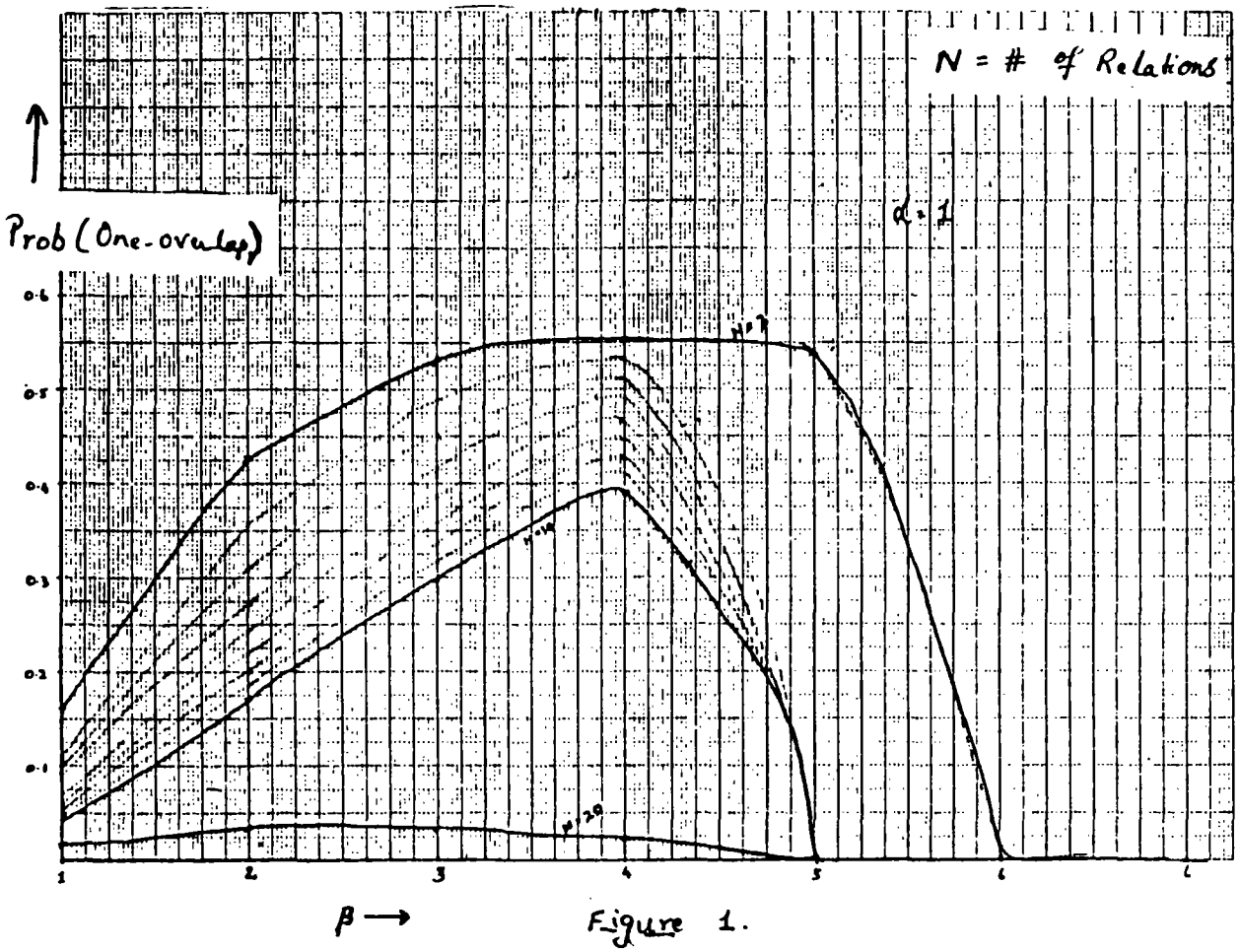


Figure 1.

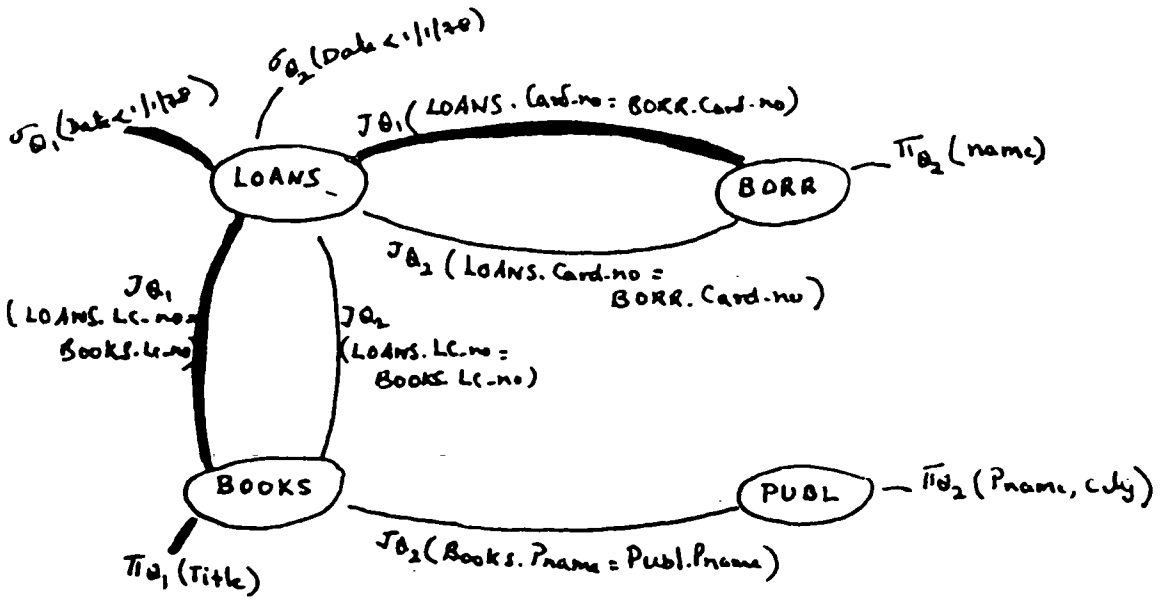


Figure 2.

REFERENCES

[CHAK82]Chakravarthy, U. S., Multiple Query Processing in Database Systems, Thesis Proposal, Dept of Computer Science, Univ of Maryland, College park, Mar 1982.

[GRAN80]Grant, John and Jack Minker, Optimization in Deductive and Conventional Relational Database Systems, pp. 195-234 in Advances in Database Theory, Vol 1, ed. H. Gallaire, J. Minker and J. M. Nicholas, Plenum Press, New York, 1980.

[KIM80]Kim, Won, Query Optimization for Relational Database Systems, Ph.D Thesis, Univ of Illinois at Urbana-Champaign, 1980.

[KING81]King, Jonathan J., Query Optimization by Semantic Reasoning, Ph.D Thesis, Dept of Computer Science, Stanford University, May 1981.

Some Thoughts on the Future Direction of Query Processing

Won Kim

IBM Research Laboratory
San Jose, California 95193

There are several areas of research in query processing which I believe will have a large impact on the performance of database systems. They include global optimization of multiple queries, optimal use of main-memory buffer space to reduce page I/Os in processing queries, and optimization of SQL-like nested queries. This paper elaborates only on these topics. However, there are a few emerging areas of research where a fresh new approach to query processing may generate some important results. One of them is the management of statistical databases [BATE82], in which a typical query requires retrieval of a large volume of data and correlation of various underlying relations (record types). Another is the logical integration of heterogeneous database systems [DAYA82], in which a query expressed in some language must be translated and optimized into a semantically equivalent query (queries) in another language.

1. Global Optimization of Queries

Optimizers in presently operational relational systems only attempt to minimize the cost of processing a single query. The cost of sequentially processing a set of n queries is simply the sum of the cost of processing each of the n queries. The sequential processing of a set of queries is usually appropriate for an online, interactive use of a database system. However, this approach may be highly inefficient for batch processing of queries embedded in conventional high-level, algorithmic programming languages. Often, it may also be inefficient for processing a set of queries which may be explicitly issued as a unit by the user in interactive mode or which may be automatically triggered to check for possible violation of integrity constraints when the user issues a data manipulation statement (update, insert, or delete).

If the cost of processing a set of queries is to be significantly lower than the sum of the cost of processing each of the queries, each subset of the queries which references the same relation(s) must be processed simultaneously when the relation(s) are fetched. A strategy which enables the simultaneous processing of a set of queries requires global knowledge of the characteristics of the queries. Global knowledge of both internal database characteristics (such as the size of relations and available access paths) and query characteristics (such as the relations and columns the query references and the expected size of the query result) makes it possible to determine not only the subsets of a given set of queries for simultaneous processing, but also a near-optimal set of secondary indexes and sorted copies of relations for processing the given set of queries.

A two-stage strategy for optimizing a set of queries and data manipulation statements based on preprocessing of queries is offered in [KIM80b] as a first-cut solution to this problem. The first (or compile-time) stage is a compile-time analysis of the query and database characteristics. It consists of two phases: Phase C.1 and Phase C.2. Phase C.1 determines an optimal set of indexes and sorted copies of relations, while Phase C.2 derives an optimal sequence of groups of que-

ries and data manipulation statements for simultaneous processing. Compile-time analysis of a given set of queries makes use of query characteristics and the access paths for processing each of the queries which a conventional single-query optimizer has determined.

The second (or run-time) stage executes the program which contains the set of queries and data manipulation statements. This stage in general must also consist of two phases: Phase R.1 and Phase R.2. Each group of queries and data manipulation statements selected for simultaneous processing is executed during Phase R.1 in the order determined during the compile-time stage. The results of the preprocessed queries are stored on the disk and the initial program is modified to replace all preprocessed queries with references to their stored results. Those queries which require unique access paths, which other queries cannot also take advantage of, may be excluded from processing during Phase R.1. During Phase R.2 the modified program is run in order to process such queries and to further process and output the results of the preprocessed queries.

During Phase R.1 the main-memory buffer is partitioned into three areas: an input area to hold the data pages that contain tuples of the relation(s) and secondary indexes on the relation(s); a program area to hold all the procedures required for processing a given set of queries simultaneously; and an output area to hold the results of the queries until they are written to the disk.

The possibility of keeping the intermediate results (or temporaries) of a query and using them in processing other 'similar' queries is explored in [FINK82]. This approach may be effective when a given set of queries consists of queries that can be processed using the results of processing other queries in the set.

2 Optimal Use of Main-Memory Buffer Space

In [KIM80a] a new method of scanning relations is presented which takes maximum advantage of available main-memory buffer space. This method, called the **nested-block method**, is superior to the nested-iteration method for computing the product of relations, and can often outperform the merge join for computing the join of relations. It is shown in [KIM81] that the method is usually also superior to the existing method of computing the division of a relation by another relation. In order to demonstrate the performance enhancement that a systematic use of main-memory buffer space may bring about, this new application of the nested-block method is illustrated here.

The binary division of a relation of degree 2, R1(C1, C2), by a unary relation, R2(C2), yields a unary relation Rt(C1). The quotient, Rt, can be obtained by grouping the dividend, R1, by the values in the C1 column, and extracting the C1 values from each group of tuples that contain in the C2 column **all** the values of R2. Since the dividend needs to be grouped by the values of the quotient column, sorting of the dividend relation has been suggested as a method for implementing the relational division. The algorithm below provides a description of the use of the nested-block method (in conjunction with hashing) for implementing the binary division of R1 (C1, C2) by R2 (C2).

Consider dividing R1 by R2, shown below. Assume that the values in the C2 column of R1 and C2 column of R2 are drawn from the same domain. That is, the C1 column of R1 is the quotient column for the division.

R1		R2
C1	C2	C2
c	x3	x1
b	x1	x2
a	x2	x2
a	x3	x3
b	x3	
a	x1	
a	x2	
b	x4	
c	x1	
b	x5	

1. Duplicates, if any, are removed from R2.
2. An initially empty list is constructed for each C2 value in the duplicate-free, R2'.
3. For each R1 tuple, if the C2 value matches a C2 value in R2', the C1 value of the R1 tuple is appended to the list for the C2 value of R2'; otherwise, the R1 tuple is discarded.
4. After R1 has been completely scanned, each of the resulting lists of C1 values is sorted and duplicates removed. The following three lists are obtained.

   x1: a, b, c
   x2: a
   x3: a, b, c

5. Those C1 values of R1 that appear in every list belongs to the quotient of the division of R1 by R2. This step in effect merge joins all the lists. This is one more reason why each of the lists is sorted on step 4. For the present example, only one value, 'a', is inserted into the quotient of the division.

The I/O cost of the nested-block method of computing the division of R1 by R2 is expected to be just P1 + P2, where P1 and P2 are the size in pages of R1 and R2, respectively. In contrast, the conventional, approach based on sorting requires 2*P1*log P1 + 2*P2*log P2 + P1 + P2, where log is to the base m when an m-way merge sort technique is used.

3. Optimizing SQL-like Nested Queries

One of the most interesting features of SQL is the nesting of query blocks to an arbitrary depth. Nesting of query blocks makes it possible to generalize a simple predicate of the form 'column operator value' to 'column operator query', 'query operator query'. Without this capability, the power of SQL is considerably restricted. In [KIM82] a query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood and their implementation in System R suffers from the use of the inefficient nested-iteration method. Alternative ways of interpreting queries which involve these types of nesting have provided the basis for the algorithms developed in [KIM82], which transform the queries to equivalent nonnested queries which existing optimizers are designed to process more efficiently. The algorithms are also

combined into a coherent strategy for completely processing a general query of arbitrary complexity.

A nested predicate may cause one of four basic types of nesting, according to whether the inner query block, Q, has in the WHERE clause a join predicate that references the relation of the outer query block and whether the column name in the SELECT clause of Q has associated with it an aggregate function (SUM, AVG, MAX, MIN, COUNT). A division predicate yields a fifth basic nesting.

One difficulty which [KIM82] does not fully address is the semantic ambiguity associated with duplicate tuples that result from evaluating the inner query blocks. Presently, in order to guarantee semantic equivalence of a nested query and its nonnested counterpart, the nonnested form of the query must be processed by first performing the projection and restriction operations on the relation corresponding to the relation referenced in the inner query block of the nested query.

Acknowledgement

Dave Reiner's meticulous comments on the first version of this article were very helpful.

References

[BATE82] Bates, D., H. Boral, and D. DeWitt A Framework for Research in Database Management for Statistical Analysis, in Proc. ACM SIGMOD Intl. Conf. on Management of Data, June 1982.

[DAYA82] Dayal, U. and N. Goodman Query Optimization for CODASYL Database Systems, in Proc. ACM SIGMOD Intl. Conf. on Management of Data, June 1982

[FINK82] Finkelstein, S. Common Expression Analysis in Database Applications, in Proc. ACM SIGMOD Intl. Conf. on Management of Data, June 1982.

[KIM80a] Kim, W. A New Way to Compute the Product and Join of Relations, in Proc. ACM SIGMOD Intl. Conf. on Management of Data, May 1980.

[KIM80b] Kim, W. Query Processing for Relational Database Systems, Ph.D. thesis, Dept. of Computer Science, University of Illinois, August 1980.

[KIM81] Kim, W. Query Optimization for Relational Database Systems, to appear in Advances in Data Base Management, Vol. 2, Heyden & Son, Inc., Philadelphia, PA.

[KIM82] Kim, W. On Optimizing an SQL-like Nested Query, ACM Trans. on Database Systems, Vol. 7, No. 3, Sept. 1982.

Issues in Query Evaluation

S. Christodoulakis

Computer Systems Research Group
University of Toronto
Toronto, Ontario, M5S 1A1
(416) 9785184

ABSTRACT

Nearly all cost analysis for query evaluation make several uniformity assumptions for modelling data base contents and data placement on devices. We describe how these assumptions often lead to upper bounds on the true costs. We also discuss batching queries to reduce the system workload, and we comment on our current interests in query evaluation in distributed data base environments.

1. Implications of Uniformity Assumptions

Most of the analytic models used in data base performance evaluation are based on the following assumptions concerning data base contents, data placement on devices and user requests: 1) The attribute values of the records of a file are uniformly distributed over the domain of values of each attribute, and attribute values of any two attributes are independent. We will call this the *uniformity and independence of attribute values in the file* assumption. 2) The likelihood that a block contains records qualifying in a query is the same for any block of the file. We will call this the *random placement* assumption. 3) The user queries in a time period are uniformly distributed over all the attribute values. We will call this the *uniformity of attribute values in queries* assumption.

These assumptions may be unrealistic in some actual data base environments. Often data bases describe populations such as the employees of an organization, the students of a university, the people under security surveillance. In contrast to the uniformity and independence of attribute values assumption, populations tend to have only a few members with extreme attribute values, and the values of their attributes are often correlated. The random placement assumption may also be unrealistic in certain environments. Consider an employee data base where new employee records are inserted at the end of the file. In this data base, records that qualify in queries asking for employees with high salaries, high responsibility levels and many years of experience are more concentrated at the begining of the file rather than being uniformly spread over the blocks of the file. Such non-uniform distributions of the qualifying records in a query over the blocks of a file may also occur in clustered files as result of correlations and other dependencies among the clustering attribute and the other attributes of the file. Finally, the assumption of uniformity of attribute values in queries may be unrealistic in certain

environments because users may be more interested in a subset of the attribute values (for example high salaries). In [CHRI81] we present evidence from actual data base environments that these assumptions often are not satisfied. However, these assumptions are easy to use in analytic models, and in some cases they are approximately satisfied.

Since these assumptions are widely used in data base performance evaluation, it is important that we understand the impact on data base performance of using these assumptions when they are not actually satisfied. We have shown that these assumptions lead to cost estimations that are often pessimistic.

Some of the results presented in [CHRI82a] are the following: The block access cost function is

$$C(P_1^Q, \cdots, P_M^Q) = I^Q + \sum_{i=1}^{M} (1-(1-P_i^Q)^n)$$

where n is the number of records qualifying in a query Q, M is the number of blocks in the file, I^Q is the cost of accessing the indices, and P_i^Q is the probability that Q accesses block i. This function has the property of being "Schur concave" [MARS79]. Schur concave functions have some important majorization properties. As a result of being Schur concave the above function acquires higher values as the probability distribution becomes less skewed, and it maximizes for a uniform distribution $P_i^Q = \frac{1}{M}$. Thus the random placement assumption is pessimistic. The difference in the cost can be high.

An analogous result has been shown for the distribution of attribute values. For uniform queries over all the values of an attribute, the average block access cost function is Schur concave with respect to the distribution of records over the attribute values. Thus the more skewed the distribution, the less the expected cost. The average cost is greatest for uniform distributions of attribute values. Since it is well known that in many actual data base environments skewed distributions of attribute values are common, this assumption is also pessimistic.

Similar results have been obtained for the independence of attribute values assumption. Uniformity and independence of attribute values have also been shown to be pessimistic when the cost function is the number of attribute values which participate in a join (following a selection).

These results have some important implications for data base performance. For performance predictors for data base designs, they suggest that large errors may be introduced by using these assumptions. For relational query optimization, these results imply that optimizers based on these assumptions will choose exhaustive strategies (like sequential scan or sorting) more often than necessary. For data base design, these results imply that exploitation of non-uniformity and dependencies of attribute values, non-random placement of qualifying records in the blocks of a file, and non-uniformity of queries could reduce the overall system cost.

2. Batching Queries

The number of users that access a common file at one time may be large. From the systems point of view, batching of requests nearly always reduces the work. Thus for overloaded sites batching of requests may be necessity rather than choice. From the on-line user's point of view, the desirability of batching depends on its effect. In order to batch a number of queries a waiting period is required. On the other hand, since the time required to process a batch of n queries is less than the time required to process the n queries individually, and because the load of the system decreases, the average response time may decrease [SHNE76]. This is more realistic in high activity environments where the waiting period is short.

Batching of requests is easier and more profitable for sequentially accessed files than for tree structures. When the levels of the tree are many the probability that more than one requests refer to the same path is small and therefore the profitability of batching decreases. Moreover, updates are difficult to batch in B-tree organizations because as result of the update other nodes in higher levels of the tree may have to be accessed [SHNE76]. Batching becomes even harder for multiattribute queries because a number of tree structures may have to be accessed and the resulting pointers to be merged.

When the number of queries batched is very large, sequential scan of the file is an alternative, possibly more profitable strategy. However, for on line environments where the number of batched requests cannot be very large without deterioration of responce times, sequential scan of a large file may be an expensive strategy.

We have investigated an alternative approach [CHRI82b]. We use an access file which is much smaller than the file itself as an access mechanism. The access file contains abstractions of the attribute values of the attributes of the file. The access file is sequentially scanned to provide pointers to the qualifying records of the file. A superset of the qualifying records is retrieved and examined for qualification before it is returned to the user. As a result of the sequential scan batching of queries becomes easy, and since the access file is small, more profitable than the sequential scan of the whole file for a moderate number of queries in the batch. Moreover, the system can make efficient use of its buffers.

In this environment the parameters of the access file design depend on the type and distribution of the user requests as well as on the data base contents. Closed form formulae for the optimal choice of parameters have been derived. We have used a similar access method in an environment where queries may refer to non-formatted data as well [TSIC82], [CHRI82c].

3. Distributed Query Processing

We are also investigating some problems in distributed data base environments. Our model is a star network environment with (partially) replicated data. Previous approaches examining query evaluation in this type of environment have not considered queueing delays introduced in the communication lines. However, queries submitted in a satellite may be processed in the satellite or in the central site depending on the communication delays due to

queueing as well as on query type, data base contents, and speed of devices. The performance model should take into account all these factors.

References

[CHRI81]

S. Christodoulakis: "Estimating Selectivities in Data Bases", Ph.D Thesis, Technical Report CSRG #136, University of Toronto, Dec. 1981.

[CHRI82a]

S. Christodoulakis: "Implications of Certain Assumptions in Data Base Performance Evaluation", submitted for publication, 1982.

[CHRI82b]

S. Christodoulakis: "Access File for Batching Single Attribute and Multiattribute Queries", in preparation, 1982.

[CHRI82c]

S. Christodoulakis and C. Faloutsos: "Performance Considerations for a Message File Server", in Alpha-Beta Technical report CSRG #143, University of Toronto, June 1982 (F. Lochovsky editor).

[SCHN76]

B. Schneiderman and V. Goodman: "Batched Searching of Sequential and Tree Structured Files", ACM TODS 1,3, March 1976, pp. 268-275.

[TSIC83]

D. Tsichritzis and S. Christodoulakis: "Message Files", ACM Transactions on Office Information Systems 1,1 Feb. 1983 (to appear).

Investigating Access Paths for Aggregates Using the "Abe" Statistical Query Facility*

Anthony Klug
Computer Science Department
University of Wisconsin
608-262-1965

1. Introduction

Since an increasingly important part of information processing today involves the taking of counts, sums, averages, and other statistical or aggregate quantities, we have been investigating the access path selection problem for these *statistical queries*. The language we have been using is the "Abe" statistical query facility being developed at the University of Wisconsin. The Abe query language is powerful, yet simple. It is a pure relational calculus language with a friendly full-screen user interface.

It turns out that the access patterns in computing a query having aggregates are virtually identical to those encountered in computing a simple join. We use three scan procedures for computing aggregates: one is like a file (or segment) scan; one is analogous to an index scan; and one is similar to a merging scan. The first two types of scans combine to form a nested loops join.

The rest of this report is divided into two parts: The first part briefly describes the Abe language, and the second part discusses Abe access paths.

2. Abe Background

Space limitations do not permit us to give an extensive description of the Abe query language. We will give a brief introduction to the language and a few examples. More examples can be found in [Klug81].

Abe (*A*ggregates *b*y *e*xample) is a domain calculus language with aggregates. It uses 2-dimensional tables as QBE does, but the methods for forming aggregates are more general, and the semantics are rather simple.

In more detail, an *Abe query* consists of a top-level query and some number of subqueries in a tree structure. A *top-level query* or a *subquery* consists of an output list, an optional condition box, and zero or more relation tables. An *output list* is simply a list of items. A *condition box* is a list of conditions, each condition being a pair of items connected by one of the operators "=", "≠", "<", "≤", ">", or "≥". A *relation table* for a relation R(A,B,C,...) is a table labeled 'R' having one or more rows and having columns labeled A,B,C,... An *item* represents a single value and is used to fill in the above tables and lists. It is one of the following four objects: a *constant* such as the number 10 or the string "joe"; a *variable* which is an identifier such as *emma* (single underlining indicates variables); or a *fixed variable* which is also an identifier such as *demo* (double underlining indicates fixed variables). Fixed variables only appear in subqueries where some parent query has an ordinary variable of the same name. Finally, an *aggregate expression* (or simply, an aggregate) is an aggregate function (count, max, min, ave, sum) followed by a column indicator (except for count) followed by the name of a subquery.

*This work was supported in part by NSF Grant MCS8102864

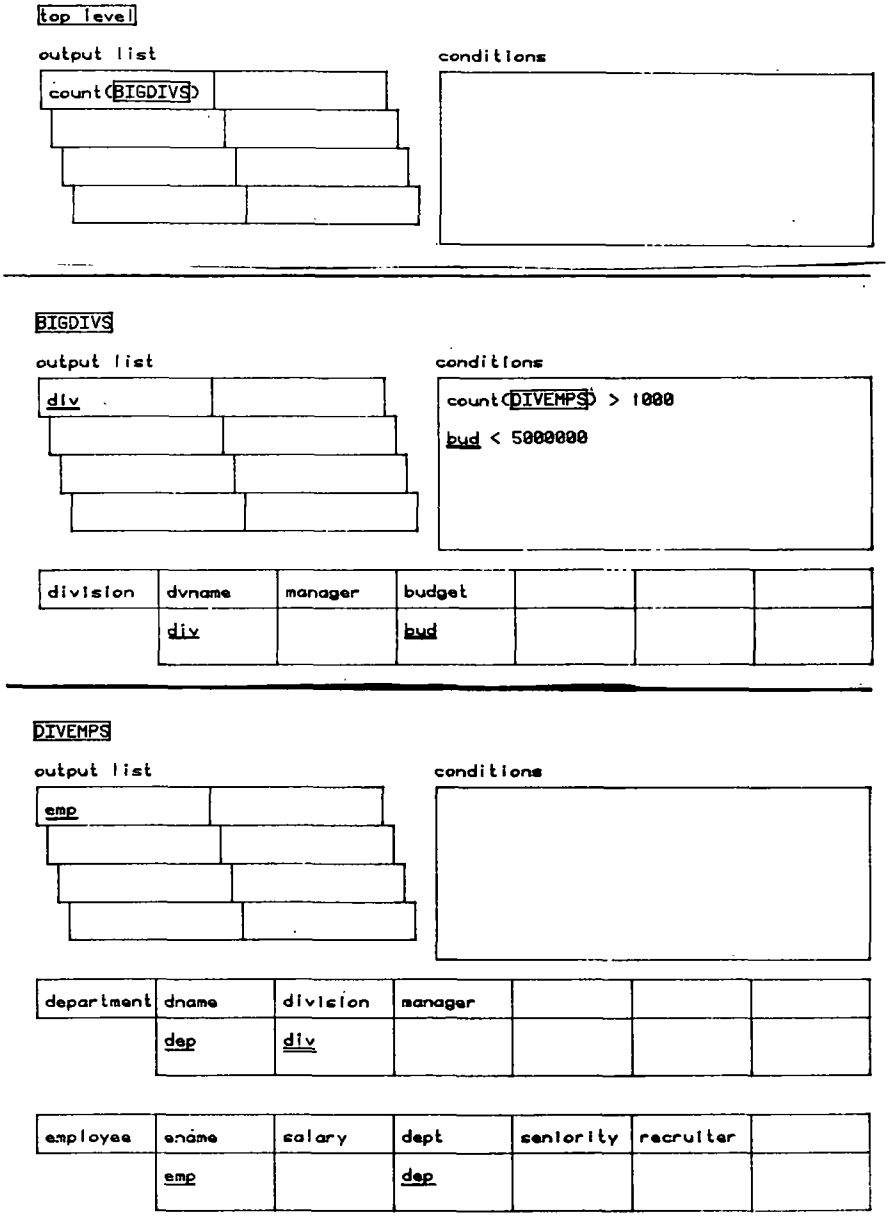
Suppose we have the following relations:

division(dvname,manager, budget)
department(dname, division, manager, budget)
employee(ename, salary, dept, seniority)

We could ask the following query:

How many divisions have more than 1000 employees and a budget less than $5000000?

This query would be expressed in Abe as in Figure 1. (each level is a separate "screen" or "window" on the terminal):



The rules for evaluating a query such as this one are briefly:

(1) Find a ***match*** for rows in the relation tables with tuples in the database. Constants match only themselves. Variables match anything subject to the condition that all occurrences of the same variable must match the same value. Fixed variable matching is described in the next paragraph.

(2) ***Evaluate*** conditions in the condition box. Evaluate any aggregates by replacing fixed variables in the subquery by the current values of the corresponding ordinary variables.

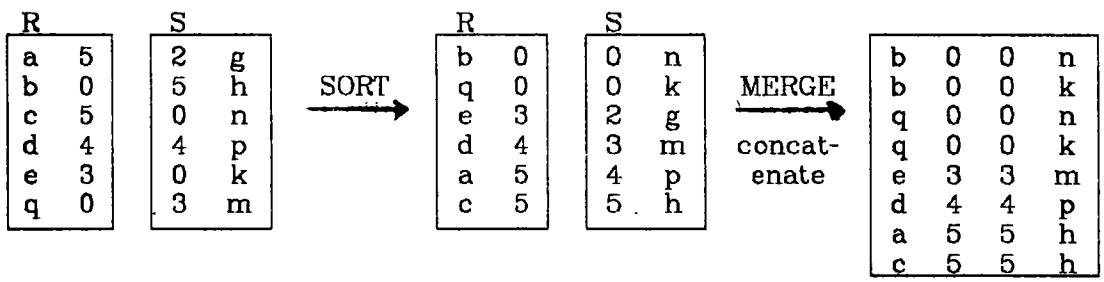
(3) If the conditions are all true, ***generate*** an output list tuple.

(4) Repeat steps (1)-(3) until no more matches are found.

3. Abe Access Paths

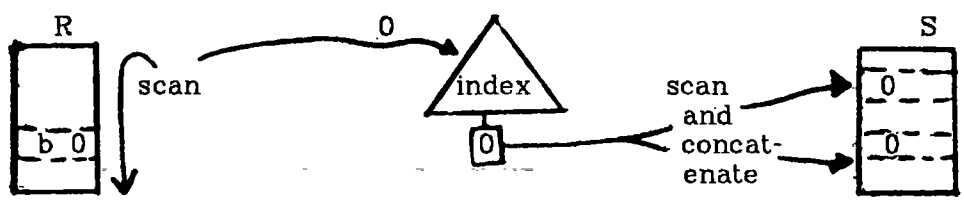
In this section we give the basic ideas involving access paths for evaluating queries with aggregates.

First consider the two main implementations for ordinary joins [SELI79]: the sort-merge and the nested loops algorithms. To join relations R and S on columns X and Y, resp., using a sort-merge, R and S would first be sorted on their join columns if necessary, and then simultaneous scans would be made on both relations, joining tuples in blocks having the same join column value:



Nested loops joins generally require an index on the inner relation. To do the above join on R and S using a nested loops algorithm with S as the inner relation, a scan is made on R, and for every qualifying tuple t in R, the X value of t is used as an entry value into the index on column Y of S. The index gives the tuple ids for all tuples in S having the same value in Y, and a scan returns these tuples. They are concatenated with t and output:

NESTED LOOPS



There are two important points to understand in applying sort-merge and nested-loops algorithms to access paths for aggregates: The first is that (at least in Abe) aggregates are almost never computed by themselves; they are always applied to some subquery and their values are used in a parent query in the condition box or the output list. Thus we can consider an aggregate compu-

tation as a binary operation of joining the subquery aggregate values to the parent query tuples. The second point is that the access patterns for computing aggregates are almost identical to those for computing ordinary joins. For example, to compute the join of department with employee, we need to, for every department tuple, access all related employee tuples and output the join tuple. To compute the query which lists departments having more than 100 employees, we need to, for every department tuple, access all related employee tuples and compute the count. Thus we have aggregate sort-merge and aggregate nested-loops algorithms. Pictures for these algorithms can be obtained from the above pictures by replacing "concatenate" by "accumulate". For example, for each department tuple, we access all related employee tuples and increment a counter for each employee tuple.

Now we will briefly discuss when an aggregate sort-merge might be better than an aggregate nested-loops, and vice versa. In general, a sort-merge is better when a high percentage of the tuples in the specified relations will participate in the output. A nested loops is better when a small percentage of the tuples in the specified relations will participate in the output. Consider the two queries (expressed in English): Q1: List all departments and their employee counts. Q2: List departments with budgets in the highest 10% of department budgets and their employee counts. If we used a nested loops to evaluate Q1, employee pages would be accessed many times, leading to an inefficient access path. If we used a sort-merge for Q2, the work of sorting the employee relation would be 90% wasted since only 1/10-th of the employee partitions would be selected. This has been known for some time for ordinary joins, but it has not been generally recognized that the same rules apply for computing aggegates.

Many details of the access procedures, their cost functions, and several examples can be found in [KLUG82].

4. Current and Future Work

We are currently working on an implementation of access path selection using these ideas. We still need to consider how ordinary joins interact with aggregate computations. In the first example of this paper, there is a subquery which is an argument to an aggregate and which itself contains a join. There may be several ways to evaluate this query efficiently.

5. References

[KLUG81] Klug A. "Abe – A Query Language for Constructing Aggregates-by-Example" Workshop on Statistical Database Management, Menlo Park, Calif., December 1981; also Univ. of Wisc. Comp. Sci. Tech. Rep. #475

[KLUG82] Klug A. "Access Paths in the 'Abe' Statistical Query Facility", Proc. ACM-SIGMOD Conference, Orlando, 1982; also Univ. of Wisc. Comp. Sci. Tech. Rep. #476

[SELI79] Selinger P., Astrahan M.M., Chamberlin D.D., Lorie R.A. and Price T.G. "Access Path Selection in a Relational Database Mangement System", ACM-SIGMOD 1979 International Conference on Management of Data

Strategy Spaces And Abstract Target Machines For Query Optimization

David Reiner and Arnon Rosenthal
Sperry Research Center
Sudbury, Massachusetts 01776
(617)-369-4000 (x353)

Abstract

Relational query optimizers generally take a two-step approach to generating a data access strategy for a query. First, they develop a set of possible join strategies (J-strategies). Second, they refine their strategies by considering computation graphs showing data movement, caching, sorting, index creation, and join implementations at the physical level (P-strategies). Each optimizer has a characteristic strategy space -- the potential J-strategies and P-strategies it considers to respond to queries.

We briefly characterize our J-strategy space [ROSE82a], and compare the J-strategy spaces of different optimizers. We then describe a virtual machine capable of supporting the P-strategies from all the optimizers we have seen (except for operations on unnormalized objects). Finally, we comment on efficiency issues in strategy space generation and searching.

Two-Level Strategy Spaces

Given a query, relational query optimizers generally take a two-step approach to data access strategy generation. First, they develop a set of possible join strategies (information combination patterns). We may view these strategies as being supported by a virtual machine called the abstract join machine. Second, perhaps interleaved with J-strategy generation, they refine their strategies by considering data movement, caching, sorting, index creation, and join implementations at the physical level. We may view the resulting computation graphs of operators (P-strategies) as being supported by a virtual machine called the abstract physical machine (P-machine). Cost models refer to the P-machine.

Each optimizer has a characteristic two-level strategy space -- the potential J-strategies and P-strategies it considers to respond to queries. The two-level view makes it easier to give fairly uniform descriptions of the set of strategies searched, explicitly or implicitly, by a given optimizer.

In [ROSE82a], we suggest implementing a query optimizer which allows a full range of P-strategies for each join in a J-strategy. Enhancements to the strategy space may occur at either level without affecting the other. Joins involving indexes can be added at the J-strategy level, for example, or a new join implementation or storage structure at the P-strategy level.

Achieving Leverage By Extending The Use Of Joins

In [ROSE82a], the basic joinable data objects are tables, collections of identically structured first normal form objects accessed by the run-time system. Under this fairly abstract definition, not only relations but also certain indexes, Codasyl set links, and other direct access structures (DAS's) may be regarded as joinable tables.

We consider a number of sophisticated DAS processing techniques, which greatly broaden the J-strategy space searched, and reduce I/O costs substantially for some queries. Two DAS's may be joined (on value or pointer fields) to produce an access list table, which no longer supports keyed access, but is a joinable entity. Thus an access list table may be joined with an underlying table. A DAS may also be joined with a foreign table (semijoin).

We recognize DAS's which contain all fields required by the query from their underlying tables, and may therefore be referenced instead of the underlying tables. Such DAS's (and access list tables) are called shadow tables.

Codasyl record types may also serve as tables. In [ROSE82b], we show how Codasyl set memberships can be represented by set-based join predicates. ([MANO82] uses a similar approach.) The link structure acts as a DAS which supports the join. This approach allows the full range of join implementations and access strategies in the P-strategy space to apply to Codasyl set processing. It is also possible to use "artificial" joins to obtain a relational view of hierarchical data stored positionally without explicit pointers (e.g. COBOL records, PL/1 structures).

By extending its concept of a join to a variety of situations where information is being combined, an optimizer can obtain more generality and flexibility for little additional cost in code and complexity.

Comparing J-Strategy Spaces Of Different Optimizers

Some factors which yield a large J-strategy space are allowing multiple temporary results to exist, considering a large number of consecutive joins before choosing which join is to be performed first, and extending joins to model sophisticated manipulations of DAS's and Codasyl set traversals. Of course, larger strategy spaces are a mixed blessing, usually requiring more time to implement, generate, and search, so there is merit in imposing restrictions on them.

We will compare the J-strategy spaces of several current optimizers. IBM's System R [SELI79] considers implicit join trees whose left subtrees represent successive accumulated temporary results, and whose right subtrees consist of single relations joined in one at a time. Multiple temporary results cannot exist. Since each left subtree always represents the outer loop of a join, and each right subtree the inner, join implementations in a System R P-strategy depend on the positions of operand tables in the parent J-strategy. In a sense, the two levels of the strategy space are not orthogonal. System R also recognizes (but does not construct) shadow tables.

The University of California's INGRES system [WONG76] picks the cheapest join to do first, processes that join, and then reexamines the remaining joins. While INGRES can in theory generate any possible tree, in fact it generates very few alternatives with its "greedy" lookahead heuristic. (Actually, the tuple substitution technique leads to a J-strategy which is acyclic but not a tree when one substitutes for a relation which participates in more than one join.) INGRES's interpretive strategy allows more accurate cost estimation than in System R, since the sizes of intermediate results are

known, but System R considers entire access strategies before committing itself to the first join. Join implementations in INGRES's P-strategies are not independent of their parent J-strategies, since a merge scan may be used only for the final join. All preceding joins are done by tuple substitution, which is equivalent to a nested loops implementation.

An interesting variation of the INGRES heuristic appears in the ADAPLEX design of Computer Corporation of America [CHAN81]. A compile-time parameter called the "optimization level" is set to a value n. The optimizer finds the cheapest n joins and performs them first, regardless of whether the joins are connected. Then all remaining sets of n joins are considered, and so on. If n=1, this is the INGRES heuristic.

Our optimizer design [ROSE82a] considers all join trees (i.e., allows multiple intermediate results of arbitrary complexity), and includes certain DAS's and access list tables as joinable entities in J-strategies, permitting semijoins and index intersections to be generated. The expanded INGRES optimizer [KOOI82] takes a similar approach to our design (the commercial version no longer uses tuple substitution), although it does not dynamically build new indexes or consider Codasyl record and set type processing.

A Simple But Powerful P-Machine

Recall that an optimizer's P-machine is the abstract physical machine which supports the operators appearing in P-strategy computation graphs. A well-designed P-machine should support a small set of primitive operators. The operators should be at a high enough level of abstraction to hide unnecessary details of data access and manipulation from the optimizer. However, they should be versatile enough and at a sufficiently low level to express a broad range of access strategies, including, if required, strategies for environments such as distributed systems, database machines, and systems where data may be stored under several different data models.

In [ROSE82a], we define a P-machine which supports four operators on tables -- scan, join, sort, and create direct access structure (create-das). Tables are collections of identically structured access records, which are first normal form data objects accessed by the run-time system. The operators take tables into tables. The operators are partial (do not apply to every physical state of every table), and some have several implementations, each with different input requirements. The join operator includes merge scan and nested loops equijoins [SELI79], and could also include other implementations such as nested blocks [KIM80]. We regard these as different implementations of the same operator, since they produce the same logical result in terms of information combination, although the sort order and other physical attributes of the result may differ.

Tables may be on disk (d), in main memory (m), or in stream state (s) (equivalent to tuple-by-tuple passage through a main memory buffer). The scan primitive, used to move tables from one location to another, has two implementations. Sequential scan moves tables to and from stream state (d-->s, m-->s, s-->d, s-->m). Transitions between disk and main memory are assumed to pass through stream state. Das-scan uses an existing index, hash table, Codasyl set link, or other DAS to bring a table into stream state (d-->s, m-->s). Here the DAS does not act as a table, but as a physical

access accelerator. There is no separate packaging of Select and Project operators as in the relational algebra, because applicable predicates and projections are applied (essentially for free) by the scan operator (duplicates are not removed). In a distributed database, a version of the scan operator would be used for site changes.

Sort sorts its operand table (s-->s), and create-das caches its operand table (s-->d, s-->m) and as a side effect creates a stream-state DAS on it. Stream state is a convenient way for operators to interface, and saves the cost of creating temporaries. In the expanded INGRES optimizer [KOOI82], sort and create-das are combined in a "Reformat" operator.

Except for manipulations of non first normal form data objects (e.g. indexes in the design of [YAO79]), the P-machine described above is capable of supporting the strategy spaces of all the optimizers we have seen [CHAN81], [KOOI82], [MAKI81], [SELI79], [WONG76].

Note that tables and the operators defined on them form an abstract data structure. Since certain DAS's may be regarded as tables, the four primitive table-manipulation operators apply to them as well as to base relations. For example, an index may be sorted on its pointer field to make access to its underlying table more sequential.

Generating And Searching Strategy Spaces -- Efficiency Issues

A P-strategy is a computation graph, a directed, acyclic graph whose input nodes represent tables available at the start of the computation, and whose interior nodes represent P-machine operators and intermediate tables. The P-strategy space can be represented by superimposing all alternative computation graphs for the query. During superimposition, intermediate tables which are equivalent with respect to future processing of the query are combined.

This formulation permits the use of standard graph searching and manipulation techniques by the optimizer. The most important of these is cost-based pruning, a form of dynamic programming which eliminates all but the cheapest P-strategy path leading to each intermediate table. This enables us to find the cheapest in a large set of P-strategies without necessarily generating them all.

Where J-strategies and P-strategies are generated in an interleaved fashion, it is possible to delay investigating P-strategies for joins which produce relatively large intermediate tables, hoping to find cheaper P-strategies involving smaller tables. Intermediate table sizes may be calculated from initial table sizes and predicate selectivities as J-strategies are constructed.

Of course, the easiest way to limit the size and complexity of the P-strategy space is to limit the size and complexity of the J-strategy space. For example, joins of DAS's with another table may be restricted or bypassed entirely, or the optimization level may be lowered in a CCA-style optimizer [CHAN81].

A number of miscellaneous heuristics may help as well. Adaptive tuning

techniques [REIN81] may be used to dynamically adjust the J- and P-strategy spaces based on the characteristics of the incoming query stream, or to adjust cost model parameters (such as the CPU/ I/O cost tradeoff) based on current resource utilization. Thresholds may be set to avoid spending more optimization time than the current best P-strategy would require to actually execute.

Summary

We have examined the common two-step approach to relational query optimization, where join strategies (J-strategies) are generated first, and then refined into computation graphs composed of data movement, caching, sorting, index creation, and join implementation operators at the physical level (P-strategies). We may view P-strategies as being supported by a virtual machine called the abstract physical machine (P-machine).

We briefly characterized our J-strategy space [ROSE82a], and compared the J-strategy spaces of different optimizers. We then described a P-machine capable of supporting the P-strategies from all the optimizers we have seen (except for operations on unnormalized objects). Finally, we commented on efficiency issues in strategy space generation and searching.

Acknowledgement

We would like to thank Dan Ries for his very useful comments and suggestions.

References

[CHAN81] A. Chan, S. Fox, K. Lin, D. Ries, "The Design of the ADAPLEX DBMS", Computer Corporation of America Technical Report CCA-09-81.

[KIM80] W. Kim, "A New Way to Compute the Product and Join of Relations", ACM SIGMOD Conf., Santa Monica, California, May 1980.

[KOOI82] R. Kooi, D. Frankforth, "Query Optimization in INGRES", Database Engineering, September 1982 (this issue).

[MAKI81] A. Makinouchi, et al, "The Optimization Strategy for Query Evaluation in RDB/V1", Proc. 7th VLDB, Cannes, September 1981.

[MANO82] F. Manola, A. Pirotte, "CQLF — A Query Language for CODASYL-Type Databases", ACM SIGMOD Conf., Orlando, Florida, June 1982.

[REIN81] D. Reiner, T. Pinkerton, "A Method for Adaptive Performance Improvement of Operating Systems", ACM SIGMETRICS Conf., Las Vegas, NV, September 1981.

[ROSE82a] A. Rosenthal, D. Reiner, "An Architecture for Query Optimization", ACM SIGMOD Conf., Orlando, Florida, June 1982.

[ROSE82b] A. Rosenthal, D. Reiner, "Querying Relational Views of Networks", COMPSAC 82 Conference, Chicago, Illinois, November 1982.

[SELI79] P. G. Selinger, et al, "Access Path Selection in a Relational Database Management System", ACM SIGMOD Conf., Boston, MA, May-June 1979.

[WONG76] E. Wong, K. Youssefi, "Decomposition - A Strategy for Query Processing", ACM TODS, 1, 3, September 1976.

[YAO79] S. B. Yao, "Optimization of Query Evaluation Algorithms", ACM TODS, 4, 2, June 1979.

Workshop Announcement On
Self-Describing Data Structures For Information Exchange

Time: October 27 & 28, 1982
Location: University of Maryland, College Park, Maryland 20742, USA
Sponsored by: Department of Computer Science, University of Maryland
NASA, Goddard Space Flight Center, Greenbelt Maryland

Program Committee: Nick Roussopoulos, (Chairman), University of Maryland
Richard desJardins, Computer Technology Associates
Edward P. Greene, NASA, Goddart Space Flight Center
John Mylopoulos, University of Toronto

The notion of a self-describing data structure is intended to be used as an information exhange protocol as well as a model for managing large bodies of knowledge about data. Some of the most important properties of a Self-Describing Data Structure (SDDS) are briefly outlined below:

An SDDS captures its description in its body. The description must be such, so that very little external (meta) knowledge is required to interpret the SDDS. Restructuring of an SDDS is done via a set of disciplinary operators which quarantee that the results of applying them on an SDDS is also an SDDS. The semantics of the results and the descriptions of the derived SDDS are inherited from the descriptions of the operands. The evolution of an SDDS is captured in its description by including its derivation, i.e. the operand(s), the operator used to derive the result SDDS, the time of derivation, and the name of the user. Thus, from the SDDS description, one can follow its derivation and the derivations of its ancestors to obtain a complete history of its existence.

A Self-Describing Model is a data description management tool for an SDDS. It maintains data descriptions and their evolution through time. This is in contrast to the conventional database models and management systems which only deal with changes in the data values but not changes in the data (schema) description. It also differs from a data dictionary system because the latter does not explicitly model time, data description evolution, and deductive mechanisms for property inheritance of these descriptions.

For the first workshop on SDDS we have solicited position papers on

models and modeling primitives of SDDS and knowledge evolution,
deductive mechanisms for property inheritance,
management tools for knowledge about data descriptions and its catalog,
techniques for dealing with multiple views of the same data and the descriptions of these multiple views,
techniques for modeling user-to-data dynamic relationships and integration of views, etc.

For more information on the workshop program contact Nick Roussopoulos, Dept. of Computer Sci., Univ. of Maryland, College Park, Md 20742, Tel:301-454-4251, 454-2001.

Call For Papers And Workshop Information
Seventh Workshop on Computer Architecture for Non-Numeric Processing

Snowbird, Utah, March 6-9, 1983

Sponsored by: ACM SIGARCH, SIGIR, and SIGMOD
Computer Science Department, University of Utah

As the costs to design, implement, and maintain a large scale system change from where hardware costs predominate to where software costs do, it is reasonable to explore computer architectures which differ from the classic numerically-oriented machine. Front end processors, performing protocol translations, are now common in data communications systems, and a variety of backend processors for database and information retrieval applications are available or currently under development. New architectures are being developed for robotics, searching and sorting, artificial intelligence, highly available systems, workstations, and text processing.

In the past, this workshop has been a primary avenue for those engaged in research and development of a variety of specialized non-numeric systems to discuss their current activities and future directions. It has proved invaluable to students conducting research in computer architecture, allowing them to present preliminary results of their work, and receive comments and suggestions from others in the field.

Registration Information

The workshop will be held at the Snowbird Ski and Summer Resort, located in Little Cottonwood Canyon near Salt Lake City, Utah. Transportation is available from the Salt Lake airport. The registration fee of $300 includes double occupancy rooms for three nights, breakfast, and the workshop banquet. The remaining meals and any skiing expenses are the responsibility of the participants. Rooms will be available for check-in at 3:00 PM Sunday, March 6, and the first workshop session will be held at 7:30 that evening.

Because of severely limited space, attendance at the workshop will be by preregistration only. Rooms have only been reserved for the workshop participants. If you wish to bring along family members, you should contact the Workshop Chairman or Snowbird as soon as possible. For those wishing to arrive on Saturday, rather than Sunday, a limited number of rooms are available, at an additional $45.

To register for the workshop, write the Workshop General Chairman by January 1, 1983, indicating your name and affiliation, mailing address and telephone number, your interest and background in computer architecture for non-numeric processing, and whether you have submitted a paper. Include the appropriate registration fee ($300 normally, $345 for early arrival). Acceptances will be sent out on January 15, 1983, and registration checks for those we are unable to accomodate will be returned. Priority on registration will be given to those submitting a paper for presentation.

Instructions for Authors

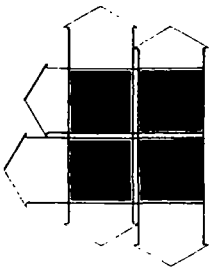
We invite papers on current or proposed work in all areas of specialized computer architecture for non-numeric applications, such as: data communications, information management and retrieval, workstations, highly available systems, robotics, artificial intelligence, searching and sorting, and text processing. Presentations will last from 30 to 45 minutes, with half the time devoted to the presentation of the paper and half to discussion. In addition, a final session, to be organized at the workshop, will consist of a number of short presentations for those wishing to present or discuss a concept, but unable to prepare a full length paper.

Authors should to submit four copies of their paper to the Program Chairman by November 15, 1982. Papers should be approximately 5000 words in length, and include a short abstract. Consideration will also be given to extended abstracts of about 1000 words; appropriate references and figures should be included. All submissions will be acknowledged and authors will be notified of acceptance by December 31, 1983. It is anticipated that the workshop proceedings will be published as a special joint issue of the newsletters of the sponsoring SIGs.

Workshop General Chairman:
Lee A. Hollaar
Department of Computer Science
University of Utah
Salt Lake City UT 84112
(801) 581-3203

Workshop Program Chairman:
Roger L. Haskin
IBM Research Laboratories, K52-282
5600 Cottle Road
San Jose CA 95193
(408) 256-6353

ANNOUNCING



The 3rd
International
Conference on

DISTRIBUTED COMPUTING SYSTEMS

Miami/Ft. Lauderdale, Florida • October 18-22, 1982

SPONSORED BY
IEEE COMPUTER SOCIETY
THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS INC

In Cooperation with
Information Processing Society of Japan (IPSJ)
Institut National de Recherche en Informatique et en Automatique (INRIA)

SCOPE

The scope of this conference encompasses the technical aspects of specifying, designing, implementing, and evaluating distributed computing systems. In such systems, there is a multiplicity of interconnected processing resources able to cooperate under system-wide control on a single problem, often with minimal reliance on centralized procedures, data, or hardware. The location of computing resources may span the spectrum from physical adjacency to geographical dispersion. The topics of interest include the following aspects of distributed computing systems:

- ☐ **SYSTEM AND HARDWARE ARCHITECTURE, INCLUDING SIMD**
- ☐ **DECENTRALIZED CONTROL, EXECUTIVES, AND OPERATING SYSTEMS**
- ☐ **DISTRIBUTED DATABASES**
- ☐ **LOGICAL AND PHYSICAL INTERCONNECTION NETWORKS**
- ☐ **SOFTWARE ENGINEERING AND PROGRAMMING LANGUAGES**
- ☐ **SURVIVABILITY, RELIABILITY, AND FAULT TOLERANCE**
- ☐ **SPECIFICATION, VERIFICATION, AND VALIDATION**
- ☐ **DESIGN METHODOLOGIES**
- ☐ **VLSI-BASED SYSTEMS**
- ☐ **ANALYSIS, MODELING, AND MEASUREMENT**
- ☐ **COMPUTER COMMUNICATION**
- ☐ **APPLICATIONS, INCLUDING SIMULATION**

The conference will include technical presentations, panel discussions, tutorials & exhibits.

General Chairman
H. J. Siegel
Purdue Univ.
School of EE
West Lafayette, IN 47907
Program Chairman
Carl G. Davis
Ballistic Missile
Defense Advanced
Technology Center
Tutorial Chairman
K. H. Kim
Univ. of South Florida
Exhibits Chairman
Edith W. Martin
Deputy Undersec. of Def. (Res. & Adv. Tech.), Pentagon, Rm 3E 114
Washington, D.C. 20301
Standing Committee Chairman
Charles R. Vick
Auburn Univ.
Awards Chairman
T. Y. Feng
Ohio State Univ.
Treasurer
Duncan H. Lawrie
Univ. Illinois - Urbana
Local Arrangements
H. Troy Nagle, Jr.
Auburn Univ.
Publications Chairman
Ben Wah
Purdue Univ.
Professional Societies Liaison
S. Diane Smith
Univ. Wisc. - Madison
Publicity Chairman
Bill Buckles
Univ. Texas - Arlington

TUTORIALS

Complementing the Conference there will be two full days set aside for tutorials. The following have been tentatively selected:

"Pragmatic View of Distributed Processing," *by Ken Thurber*
"Microcomputer Networks," *by Harvey Freeman*
"Fault-Tolerant Computing," *by Vic Nelson and Bill Carroll*
"Decentralized Control," *by Bob Larson, Paul McEntire and John O'Reilley*

PROGRAM COMMITTEE

Bob Arnold (Honeywell)
Geneva Belford (U. Ill.)
Bill Carroll (U. Texas-Arlington)
Glenn Cox (General Research Corp.)
Barry Gilbert (Mayo Clinic)
J. C. Huang (U. of Houston)
Robert Keller (U. Utah)
Annette Krygiel (Defense Mapping Agency)
Bill McDonald (Systems Development Corp.)
Vic Nelson (Auburn U.)
Peter Ng (U. Missouri)
Dan Siewiorek (Carnegie-Mellon U.)
Harold Stone (U. Mass.)
Ken Thurber (Architecture Tech. Corp.)
Larry Wittie (SUNY/Buff.)
Ken Batcher (Goodyear)
Bharat Bhargava (U. Pittsburgh)
Doug DeGroot (IBM)
Clarence Giese (Dept. of Army AIRMICS)
Bob Heath (U. Kentucky)
Lana Kartashev (U. Nebraska)
Jack Lipovski (U. Texas-Austin)
Mike Liu (Ohio State U.)
John Musa (Bell Labs)
Chong Nam (Systems Control, Inc.)
C. V. Ramamoorthy (UC/Berkeley)
Azriel Rosenfeld (U. Maryland)
Steve Smoliar (Schlumberger-Doll Research
Joe Urban (U. Southwestern La.)

International Associate Chairpeople

Helmut Kerner, Austria
C. M. Woodside, Canada
Paul Pearson, England
Gerard Le Lann, France
Herbert Weber, Germany
Mariagiovanna Sami, Italy
Hideo Aiso, Japan
J. Wilmink, Netherlands
R. C. T. Lee, Taiwan
Leah J. Siegel, U.S.A.

Additional Program Committee Members will be chosen by the International Associate Chairpeople.

CONFERENCE LOCATION

Diplomat Hotel in Hollywood, Florida near the international airport at Miami. Beachfront resort with tennis, golf, swimming, shopping, and boating.

If you wish to receive a copy of the Advance Program for the Third International Conference on Distributed Computing Systems, clip and mail this coupon to: Harry Hayman, IEEE Computer Society, 1109 Spring Street, Suite 201, Silver Spring, MD 20910.

NAME _____ EMPLOYER _____

STREET _____

CITY _____ STATE _____ ZIP _____ COUNTRY _____