An Iterative Method for Distributed Database Design

Rex Blankinship
Alan R. Hevner
S. Bing Yao
Information Systems
College of Business and Management
University of Maryland
College Park, MD 20742

Abstract

The development of a distributed database system requires effective solutions to many complex and interrelated design problems. The cost dependencies between query optimization and data allocation on distributed systems are well recognized but little understood. We investigate these dependencies by proposing and analyzing an iterative heuristic which provides an integrated solution to the query optimization and data allocation problems. The optimization heuristic iterates between finding minimum cost query strategies and minimum cost data allocations until a local minimum for the combined problem is found. A search from convergence efficiently scans the optimization search space for lower cost solutions. Parametric studies within a simple query environment demonstrate near-optimal performance for the iterative method when minimizing total time and response cost of queries. The iterative method provides clear improvements over alternative solution methods. The paper concludes with the practical implications of this research and its future directions.

1. Design Optimization in Distributed Database Systems

Increasingly, organizations are interconnecting computers for cooperative processing, and utilizing distributed database systems to control access to their decentralized information resources. The development of a distributed database system requires effective solutions to many complex and interrelated design issues, including network topology, hardware allocation, data partitioning, data allocation, query optimization, data replication, concurrency control, reliability, and recovery [Ozsu and Valduriez 1991]. In order to most effectively utilize distributed database systems, organizations need practical design methods which can integrate multiple design issues to achieve efficient overall system performance.

While much research has been conducted on individual distributed design problems, little progress has been made toward integrating these problems. Most of the individual design problems are NP-hard, so researchers have usually studied them in isolation to control complexity and tractability. While this approach has led to effective solutions to parts of the overall system design, the interdependencies between individual problems are still not well understood.

Query optimization and data allocation are two important distributed systems design problems that are closely interrelated. Distributed query optimization depends on how the data are allocated, since processing schedules often include operations on different sites and data transmissions between them. On the other hand, the optimal method of allocating data depends on the processing strategies used for solving queries. Typically, researchers studying data allocation assume a fixed query optimization method to generate processing schedules; while researchers on query optimization assume a fixed data allocation. By assuming a solution to one problem and solving the other, researchers control the complexity of these two problems, but fail to integrate their solutions.

Comprehensive surveys of state-of-the-art research on distributed query optimization (e.g., [Yu and Chang 1984, Hevner and Yao 1987]) and distributed data allocation (e.g., [Dowdy and Foster 1982, Hevner and Rao 1988]) exist. The majority of research in one area has assumed a given solution for the other. Only a few researchers have investigated the inherent dependencies of the two design problems. Early research by Loomis and Popek [Loomis and Popek 1976] provides guidelines for data replication and allocation based on optimizing query strategies. They point out that multiple copies of data should be placed on a network to maximize parallel processing within queries. In [Wah and Lien 1985], the authors analyze the interdependencies among data partitioning, data allocation, query optimization, concurrency control, and network design in local multi-access distributed systems. A broadcast protocol is proposed to promote sharing of information to support the integrated solution of these control problems. No specific integrated solution methods are detailed, however.

Apers develops a distributed data allocation algorithm that utilizes actual query processing schedules [Apers 1982, Apers 1988]. A virtual network is defined with each database relation assigned to a different virtual site with no relations at query sites.

Distributed query optimization is performed to generate processing schedules. An optimal data allocation is found by merging virtual data sites into actual network sites to minimize intermediate, relation-to-relation transmission costs and final, result-to-query site transmission costs. Thus, Apers' method integrates the two problems during design by sequentially optimizing query strategies and then data allocation. During execution of the distributed system, query optimization will be performed based upon the determined data allocation. Apers proposes an extended 'dynamic heuristic' to achieve greater integration of the two problems during system design. However, the approach becomes quickly intractable as problem size increases.

Sacca and Wiederhold extend Apers' approach for data allocation in systems of tightly clustered processors [Sacca and Wiederhold 1985]. Their allocation model recognizes implicit dependencies from partitions of a data entity as well as dependencies based upon user access patterns. Storage constraints of sites are also considered. The allocation process iterates between query optimization and data allocation based on a pairwise combination of data partitions at single processors in the cluster. This approach is shown to be effective on tightly-coupled processors with small communications delay. An extension of the approach to general networks is not demonstrated.

A methodology for distributed database design proposed by Mukkamala includes an iterative integration of complex design problems [Mukkamala et al. 1988]. The methodology consists of a sequential application of algorithms to optimize relation partitioning, data allocation, query optimization, and load balancing. Design evaluation, guided by an expert system, indicates when further iterations are needed to meet design goals. An internal repeating loop is shown between the data allocation and query optimization algorithms. However, no specific details on the implementation of this iterative process are provided.

In this paper, we present an iterative method for integrating realistic query optimization and data allocation methods in distributed database design. In section 2, we describe an iterative heuristic method and discuss its flexibility and power. State-of-the-art query optimization and data allocation algorithms can be 'plugged' directly into the heuristic. In section 3, we demonstrate the use of the iterative heuristic in a 'simple query' environment. Cost models are developed to demonstrate the application of the heuristic to minimize query total times and to minimize query response times. A simple example demonstrates how an 'optimal' data allocation solution can be significantly improved by integration with the query optimization problem. We have implemented selected query optimization and data allocation algorithms to perform experimental studies, as presented in section 4. Test results show the iterative method always outperforms alternative methods on total time and response time problems, yielding results that are very close to optimal. Finally, in section 5, we present conclusions and our future research directions.

2. An Iterative Heuristic Method for Distributed Database Design

The combined distributed query optimization/data allocation problem has an immense search space for optimal solution. Both optimization problems individually have been proven NP-hard. Eswaran proves that simple models of distributed data allocation are NP-hard [Eswaran 1974], and distributed query optimization has been shown NP-hard even in restricted query environments [Gavish and Segev 1986]. Thus, the combined problem is NP-hard since a non-NP solution for the combined problem would imply a non-NP solution for each of the subproblems. A more complete analysis of the search space for the combined problem is found in [Blankinship 1991].

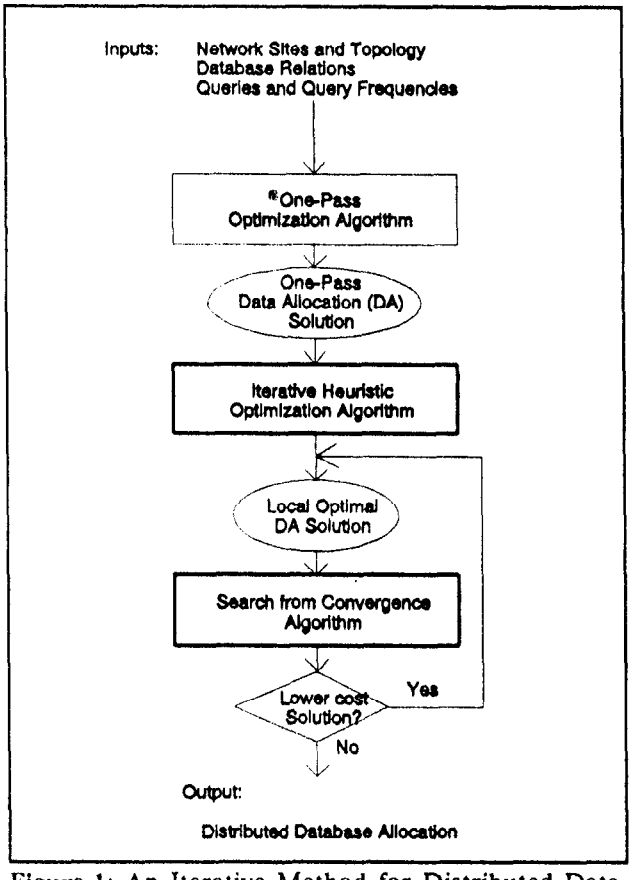

Figure 1: An Iterative Method for Distributed Database Design

Our objective is to develop a tractable heuristic that integrates query optimization directly to determine a close-to-optimal distributed database design. This research extends the work of Apers, Sacca and Wiederhold, and Mukkamala by elaborating the design

and implementation of an effective iterative heuristic for general distributed system environments. In this section, we present the design of the heuristic algorithm and discuss its power and flexibility. In Section 3, this algorithm is implemented in a simple query environment with appropriate cost models for query optimization and data allocation.

Figure 1 presents the overall heuristic algorithm. The input information consists of network sites (including local processing costs, storage costs, etc.), network topology (including transmission costs, etc.), database units of allocation (e.g., relations), and database queries (including query frequencies). (The optimization problems of data partitioning and data redundancy are not considered in this paper, but are proposed research extensions. Three distinct subalgorithms can be identified. An initial data allocation is determined via a 'one-pass' algorithm. This starting data allocation is input to an iterative heuristic algorithm that generates a local optimal data allocation solution. Then a thorough 'search from convergence' algorithm explores the problem search space to discover lower cost local minimums. The iterative heuristic is also embedded as part of the search from convergence algorithm.

2.1 One-Pass Optimization Algorithm

An initial data allocation is developed in order to 'prime the pump' for the remainder of the iterative design method. The purpose of the one-pass algorithm is to produce, in an efficient manner, a good quality starting data allocation. As described in the next section, a straightforward Most Frequently Accessed (MFA) algorithm or the Apers' one-pass algorithm [Apers 1988] can be used.

2.2 Iterative Heuristic Optimization Algorithm

This algorithm accepts the one-pass solution as a given data allocation. The heuristic then alternates between distributed query optimization and distributed data allocation optimization until a local optimum is reached. A local optimum occurs when: 1) given the existing query schedules, the data allocation algorithm cannot find a lower cost solution, and 2) given the existing data allocation, the query optimization algorithm cannot find lower cost query schedules. (Since the iterative heuristic is greedy in nature, a global optimal solution cannot be assured.) Figure 2 shows the steps of the algorithm.

The iterative approach controls the overall complexity of the combined problem. Each iteration contains a well-defined sequence of query optimization followed by data allocation optimization. The number of iterations required to find a local optimal solution is easily seen to be bounded in the worst case [Blankinship 1991]. For any reasonable distributed system cost model, a given data allocation would produce a finite cost of query processing. Since the iterative

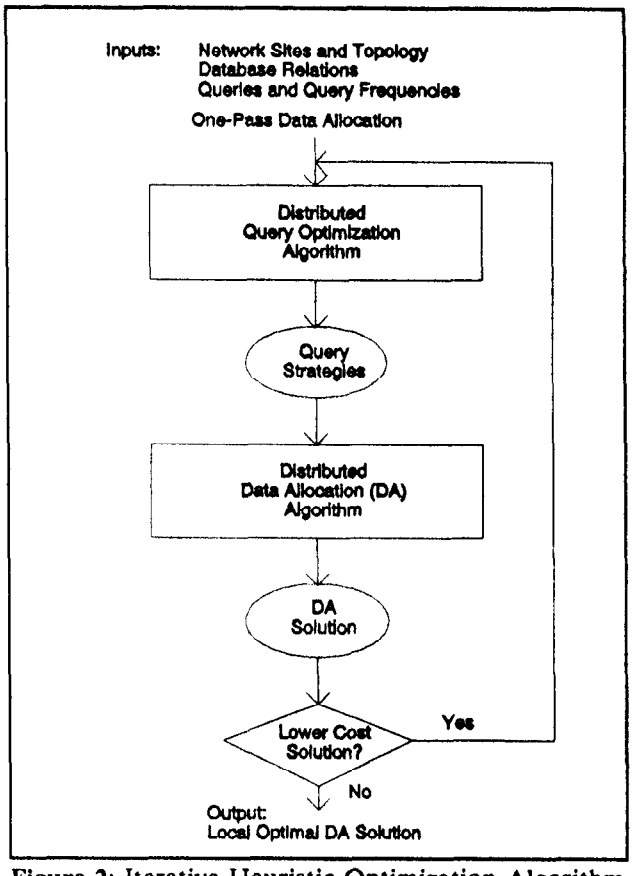

Figure 2: Iterative Heuristic Optimization Algorithm

heuristic continues only while reduced-cost data allocations are found, the number of iterations must be finite. In our experimentation with the heuristic, only in very rare cases is a local optimum not found within 2 to 4 iterations.

This iterative approach allows great flexibility in use of distributed system cost models, optimization objectives, and optimization algorithms. However, it is important that the optimization objectives be combined into a single cost equation, so that different data allocations can be unambiguously compared. Both the data allocation and query optimization algorithms should work toward minimizing this cost equation. Integrated cost models may consider query total time, query response time, local processing costs, storage constraints, load balancing over sites, etc. With sufficient insight into the specific distributed system requirements, a database designer should be able to adapt a wide range of potential query optimization and data allocation algorithms into the iterative heuristic.

2.3 Search from Convergence Optimization Algorithm

Experimentation has shown that the first local optimal data allocation may not provide a solution that is close to the global optimal data allocation. The Search from Convergence heuristic, shown in Figure 3, allows

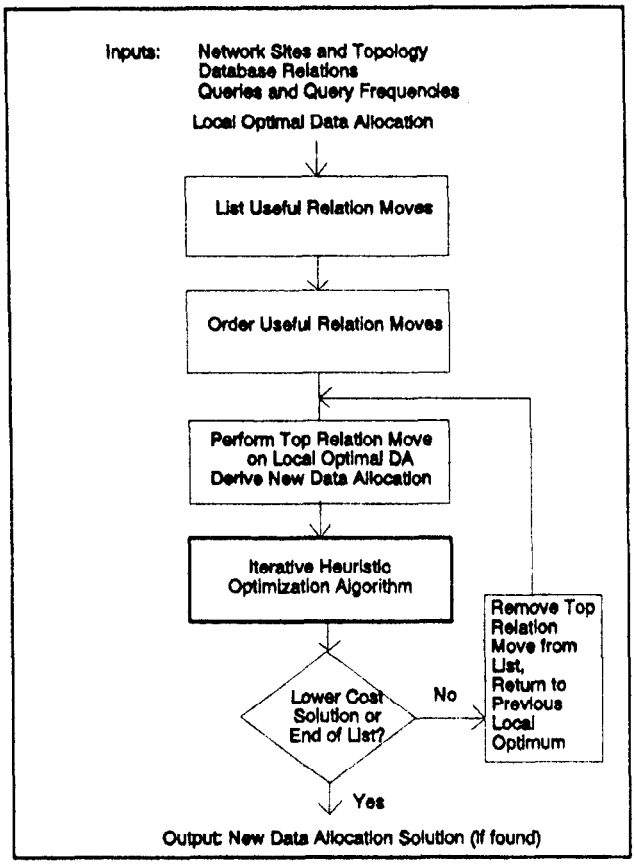

Figure 3: Search from Convergence Algorithm

the search for the global optimum to continue by discovering a new data allocation with a lower cost. A list of candidate single relation moves is defined and ordered, based on potential benefit. The method of ordering these moves for our experiments is specified in the next section. Again, the designer has the flexibility to develop a customized ordering.

For each single relation move, in order of potential benefit, a new data allocation is derived. The Iterative Heuristic algorithm is called to find a new local minimum design solution. If the new local minimum has lower cost, then we restart the Search from Convergence from that solution. If no lower cost solution is found, then the overall process is complete and the current data allocation solution becomes the final distributed database design. The efficiency of each execution of the Search from Convergence algorithm depends on the number of relations, r, and network sites, s, in the system. In worst case, r*(s-1) single relation moves would need to be tested. However, the designer can improve efficiency by limiting the search to only the n top relation moves in the list.

3. Design of the Iterative Optimization Method in the Simple Query Environment

We select the simple query environment upon which to implement, experiment, and analyze the iterative database design method. The simple query environment, as defined in [Hevner and Yao 1979], assumes a fully connected, geographically distributed network. Transmission cost equations are identical between any two sites, and costs are based on the amount of data transmitted. Local processing costs are negligible and there are no storage constraints at sites.

The relational data model is used to describe the data and query processing on the data. Only simple queries are processed. In a simple relational query, after local processing (i.e., selections, projections, and joins between relations located at a site), each query site has a single relation containing a single, common join attribute. Query optimization derives a strategy for transmitting and joining these relations in order to minimize query total time or query response time. (We note that other researchers have termed such queries 'set queries' [Gavish and Segev 1986].)

The simple query environment is chosen because it has a manageable complexity while remaining realistic and interesting. Optimization algorithms from the simple query environment have been extended to general query environments as the bases for effective heuristics (e.g., [Hevner and Yao 1979]). The parameters describing the simple query environment are:

S_i: Network sites, i = 1, 2,..., s.

For each relation R_j, j = 1, 2,..., r;

n_j: number of tuples,

a_j: number of attributes,

s_j: size (e.g., in bytes).

For each attribute d_{jk}, k = 1, 2,..., a_j of relation R_j;

p_{jk}: attribute density, i.e., the number of different values in the current state of the attribute divided by the number of possible attribute values. During join operations the density is used as a selectivity coefficient.

w_{jk}: size (e.g., in bytes) of the data item in attribute d_{jk},

For each query Q_n, n = 1, 2,..., q;

$freq_{ni}$: frequency of query Q_n entered at result site S_i during a given time unit,

$relq_{jn}$: relation R_j is in query Q_n, zero-one variable.

In the remainder of this section, we design an implementation for each algorithm in the iterative optimization method.

3.1 One-Pass Algorithm

This algorithm produces a good initial data allocation to prime the iterative heuristic. We have implemented two one-pass allocation algorithms; the MFA algorithm and Apers' static algorithm.

3.1.1 Most Frequently Accessed (MFA) Allocation Algorithm

Most existing research on data allocation assumes relations are accessed independently. Although many diverse design issues have been considered (e.g., storage capacity constraints, local processing costs, multiple copy update costs), the intermediate transmissions between relations (i.e., joins and semi-joins) which are often part of optimized query schedules, are usually ignored. To represent this approach, we implement a straightforward algorithm that assigns a relation to the site from which it is most requested; no intermediate data transmissions are considered. Thus, relation R_j is allocated to site S_i where:

$$\underset{i}{MAX} \sum_{n=1}^{q} relq_{jn}\, freq_{ni}$$

3.1.2 Apers' Data Allocation Algorithm

The Apers' static data allocation algorithm optimizes queries by assuming that each relation is located at a separate, virtual site with no relations allocated to query sites. Then the query processing algorithm is applied to obtain a strategy of transmissions for each query. The transmissions from each query strategy are multiplied by the frequency of that query. Transmissions between each pair of relations and transmissions between relations and sites are aggregated. We use the notation:

RS_{ji} = Sum of transmissions between relation R_j and site S_i, for all queries.

RR_{jk} = Sum of transmissions between relation R_j and relation R_k, for all queries.

The allocation process begins by allocating each relation to the site where it has highest traffic. Relation R_j is allocated to site S_i where $\underset{i}{MAX}\ RS_{ji}$.

While there are more relation pairs to consider, the following processing is performed:

- Select the relation pair, (R_j, R_k) with $\underset{jk}{MAX}\ RR_{jk}$.
- Calculate the net result from co-locating the relation pair:
  1) Remove the relations from the sites to which they are assigned. This increases total transmissions by $\underset{i}{MAX}\ RS_{ji} + \max_{l} RS_{kl}$.
  2) Unite the two relations and assign them to the site where the pair has highest transmissions. This decreases the total transmissions by $RR_{jk} + \max_{i}(RS_{ji} + RS_{ki})$.

  The cost savings of the co-location is the difference of these two amounts.
- If co-locating the relations is advantageous, the pair is merged. For the remainder of this algorithm, the merged pair is treated as a single relation and all transmission volumes are recalculated to reflect the merge.

When no further beneficial pairings exist, the algorithm outputs the final data allocation.

3.2 Iterative Heuristic Algorithm

The Iterative Heuristic algorithm requires the implementation of compatible algorithms for query optimization and data allocation optimization. In the simple query environment, we have previously developed optimal query optimization algorithms for total time (Algorithm Serial) and response time (Algorithm Parallel) optimization [Hevner and Yao 1979]. Here we define compatible data allocation cost models for total time and response time optimization. In [Blankinship 1991], these cost models are shown to produce optimal data allocations via exhaustive search.

3.2.1 Total Time Optimization Algorithms

Algorithm Serial produces an optimal query strategy by moving the relation with the smallest selectivity (i.e., attribute density) to the relation with the next smallest selectivity, and so on, until all relations are linked in a serial transmission pattern. When a query relation resides at the query site, two cases are tested: the regular serial pattern, and a pattern that leaves the relation at the query site out. The lower cost case is selected as optimal.

We define a data allocation cost model which incorporates the intermediate data transmissions from relation-to-relation and the final relation-to-query-site transmission. Relations are allocated to minimize the total time of these transmissions.

$$\text{Minimize} \sum_{i=1}^{s}\sum_{j=1}^{r}(1 - X_{ij})\,RS_{ji} + \sum_{j=1}^{r}\sum_{k=1}^{r}(1 - Y_{jk})\,RR_{jk}$$

where:

X_{ij} = 1 if relation R_j is allocated to site S_i, otherwise 0.

Y_{jk} = 1 if relation R_j and relation R_k are allocated to the same site, otherwise 0.

For all R_j, $\sum_{i=1}^{s} X_{ij} = 1$. (Allocate 1 copy of each relation.)

To illustrate the iterative heuristic for total time minimization, consider the following example in a simple query environment. Assume a network with three sites (1, 2, 3) and three relations (A, B, C) with the following selectivities and sizes:

Relation	Selectivity	Size
A	1.00	1000
B	.99	990
C	.98	980

There are five queries with result sites and frequencies as follows:

Query	Query Site	Query Frequency
Join A, B, and C	1	1.00
Select * from A	2	2.00
Select * from B	1	1.97
Select * from B	3	1.98
Select * from C	2	2.00

Initially, we assume all relations are at virtual sites, with no two relations at the same site, and apply the query optimization algorithm to obtain initial processing strategies. For the first query, joining relation A, B, and C, Algorithm Serial first orders the relations by increasing selectivity (i.e. C, B, A), and then performs transmissions and joins in this order. Relation C has size 980 and is transmitted to B (RR_{CB} = 980). C is joined with B, with the result relation having size 990 x .98 = 970, and selectivity .99 x .98 = .97. This result relation is transmitted to relation A (RR_{BA} = 970), and is joined with A, to produce the final result, which has size 1000 x .97 = 970, and selectivity 1.00 x .97 = .97. The final transmission delivers this result to the query site (RS_{A1} = 970).

Transmissions for the single relation queries are:
RS_{A2} = 1000 (size of A) x 2.00 (frequency) = 2000;
RS_{B1} = 990 (size of B) x 1.97 (frequency) = 1950;
RS_{B3} = 990 (size of B) x 1.98 (frequency) = 1960;
RS_{C2} = 980 (size of C) x 2.00 (frequency) = 1960.

Using these query transmissions, the data allocation algorithm can now find an initial solution. For this small example, we use exhaustive search to find an optimal data allocation. There are 3^3 = 27 ways to allocate the 3 relations on 3 sites. The lowest cost allocation is A at 2, B at 3, C at 2, with total transmission time of 4870.

Figure 4 represents this data allocation, and the initial query processing strategies. The numbers in boxes are volumes for local queries, where the allocation has made transmissions unnecessary. The solid arrow represents the query on relation B from site 1, while the open arrows represent the processing for the query on all three relations. This data allocation result could be produced by a one-pass optimization algorithm (e.g. Apers) and is optimal, given the existing query strategies.

We continue with another iteration of query optimization and data allocation to see if a better solution can be found. Since relations C and A are at the same site, Algorithm Serial initially joins these two relations. The result is then sent to B's site; the final result is sent to the query site. This solution is shown in Figure 5. The data transmissions for this new strategy are RR_{CA} = 980, RR_{AB} = 980, RS_{B1} = 970. Transmissions for the single relation queries are unchanged. Thus, the total transmission time of this combined

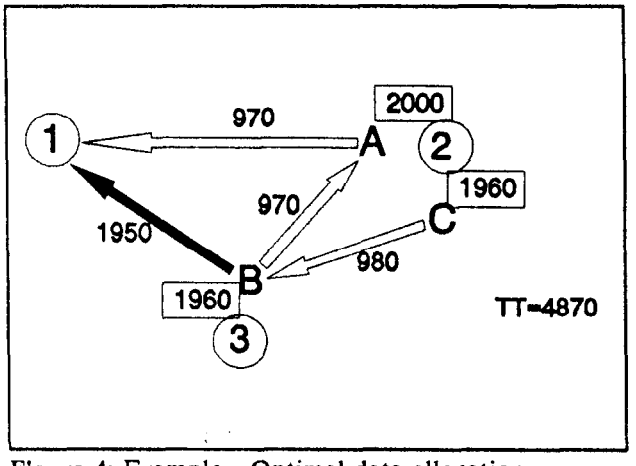

Figure 4: Example - Optimal data allocation

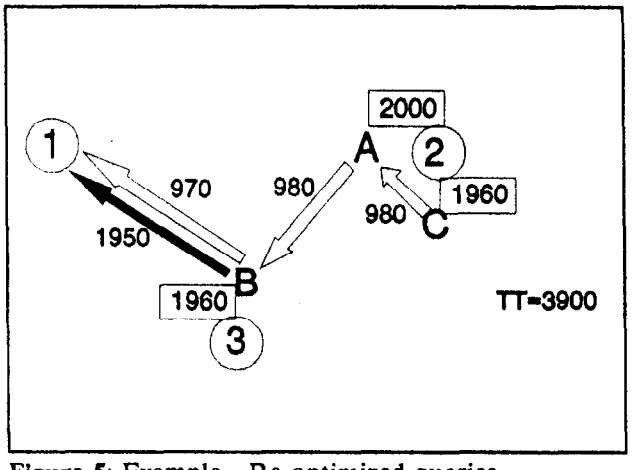

Figure 5: Example - Re-optimized queries

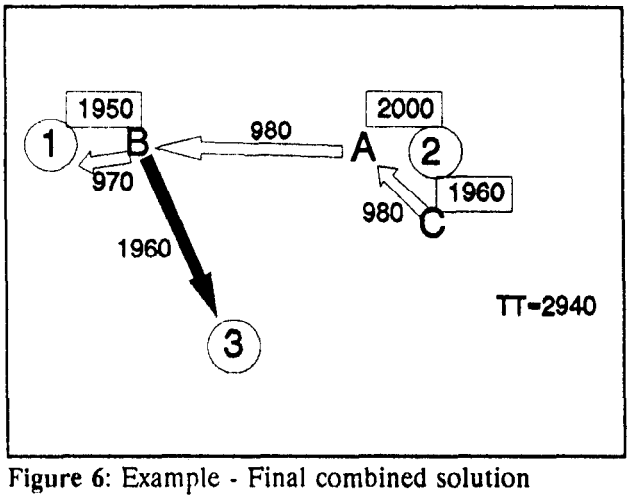

Figure 6: Example - Final combined solution

solution is 980+970+1950 = 3900. These query strategies are optimal, given the existing data allocation.

The data allocation is re-optimized by moving relation B from site 3 to site 1. The total time for this

combined solution is 1960 + 980 = 2940 (see Figure 6). Applying another iteration to this data allocation results in no change to the query strategies. Thus, the iterative heuristic has converged. This simple example demonstrates how an optimal one-pass solution can be significantly improved (40% cost reduction) by applying the iterative design approach.

Performing exhaustive search based on the above cost model is tractable only for small problems. For larger problems in our experiments, we implement Algorithm Serial and the Apers' static algorithm in the iteration to minimize total time costs.

3.2.2 Response Time Optimization Algorithms

Algorithm Parallel finds an optimal query strategy to minimize response time by emphasizing parallel data transmissions on the network. (The details of this algorithm are found in [Hevner and Yao 1979] and are not repeated here.) An original distributed cost model is derived to be compatible with Algorithm Parallel. The response time data allocation cost model is defined as follows:

S_n = Site where query Q_n originates.

TS_n = Number of (parallel) transmissions into S_n for query Q_n.

QRS_{jn} = Transmission volume for the query Q_n strategy, per unit time, between relation R_j and S_n.

TR_{nj} = Number of (parallel) transmissions query Q_n has into relation R_j.

QRR_{nkj} = Transmission volume for the query Q_n strategy, per unit time, from relation R_k to relation R_j.

The problem of allocating relations on the network to minimize response time is to minimize:

$$\sum_{n=1}^{q} \underset{j=1}{\overset{TS_n}{MAX}} [(1-X_{jS_n})QRS_{jn} + RTime_{nj}]$$

Minimize the response time total over all queries. The response time of query q is the MAX response time of the parallel transmissions into the query site, S_n. The response time of a transmission is the sum of:

1) the transmission time for the relation, R_j, to the query site, S_n (this is zero if R_j is allocated at S_n);
2) the response time of the schedule coming into relation R_j,

where:

$$RTime_{nj} = \underset{k=1}{\overset{TR_{nj}}{MAX}} [(1-Y_{jk})QRR_{nkj} + RTime_{nk}]$$

The response time of the transmission schedule into relation R_j is the MAX response time of the individual parallel transmissions into R_j. The response time of an individual transmission of relation R_k into relation R_j is the sum of:

1) the transmission time of the relation R_k into R_j, which is zero if R_j and R_k are allocated at the same site;
2) the response time of the transmission schedule into R_k, where:

X_{ji} = 1 if relation R_j is allocated to S_i, otherwise 0.
Y_{jk} = 1 if relations R_j and R_k are allocated to the same site, otherwise 0.

For each relation R_j, $\sum_{i=1}^{s} X_{ij} = 1$.

(Allocate 1 copy of each relation.)

To better understand the objective function of this cost model, consider Figure 7, which displays an arbitrary general query schedule, showing the schedule of transmissions and joins into the query site. For this query, the data allocation objective function is to minimize:

$$(1-X_{A1})1000$$
$$MAX\{(1-X_{B1})400 + (1-Y_{CB})500 + MAX\{\begin{matrix}(1-Y_{DC})150 \\ (1-Y_{EC})50\end{matrix}$$

The objective function and the query have similar form. The objective function shows recursion; the response time of relation B into the query site includes the response time of relation C into relation B, which includes the maximum of D into C and E into C. Each relation-site transmission is eliminated if the referenced relation is allocated to the referenced site; each relation-relation transmission is eliminated if both relations are allocated to the same site.

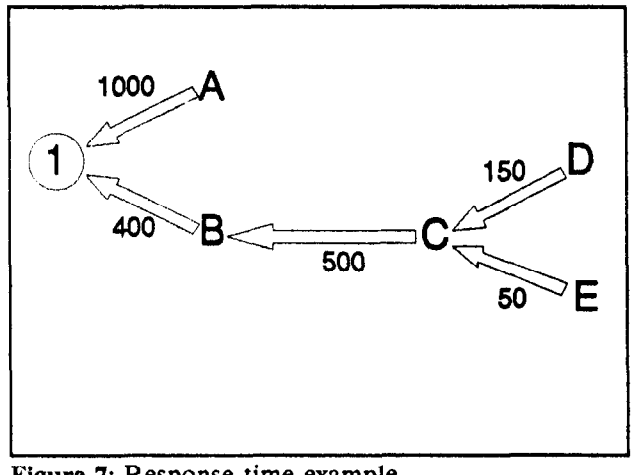

Figure 7: Response time example

Again, small problems can be solved iteratively using Algorithm Parallel and an exhaustive search for optimal data allocations using the derived cost model above. However, for larger problems, a tractable heuristic data allocation algorithm that minimizes response time cost is needed in the iteration. We have found no published algorithm to do this, so in our experiments we have modified the Apers' static algorithm for this purpose.

3.3 Search from Convergence Algorithm

When a local optimum is reached, finding a lower cost solution requires changing both the data allocation and the query processing schedules. To select and order single-relation changes to the data allocation, we analyze the "possible" query transmission patterns, instead of considering only the existing query schedules. This approach recognizes the interdependencies between the data allocation and query optimization problems, and the characteristics of local optimum solutions.

For each relation in a query, we record one possible transmission between the relation and the site of every other relation in the query, and an additional transmission to the query site. For example, if we have a query at site 1 which accesses relations A at site 2, B at site 3, and C at site 4, possible transmissions are: A to 3; A to 4; A to 1; B to 2; B to 4; B to 1; C to 2; C to 3; C to 1. For a query Q_n on r_n relations, there are r_n^2 possible transmissions. Since the number of relations in a realistic query is limited, this number is manageable. Each possible transmission is assigned the frequency of it's query. We aggregate over all queries, defining PRS_{ji} as the sum of possible transmissions between relation R_j and site S_i.

For each relation R_j, currently allocated to site S_m, we calculate the ratio:

$$\frac{\underset{i \neq m}{MAX}\ PRS_{ji}}{PRS_{jm}}$$

Relations are ordered by descending value of this ratio. The sites for each relation are ordered by descending possible transmission volume. The Search from Convergence algorithm tests the first two site moves for each relation. Thus, the maximum number of iterations is 2r.

4. Experimental Results

A major benefit of testing within the simple query environment is that optimal response time and total time query schedules can be found in polynomial time [Hevner and Yao 1979]. Testing with optimal solutions within this simplified environment allows the development and understanding of concepts that can be extended to the general query environment. Given a data allocation, it is possible to find an optimal set of query strategies by applying either Algorithm Serial (to minimize total time) or Algorithm Parallel (to minimize response time). Given a set of query strategies, it is possible to find an optimal data allocation by performing exhaustive search using the data allocation cost models of Section 3. By solving for optimal query strategies on each possible data allocation, an optimal solution to the combined query optimization and data allocation problem can be found. We use exhaustive search to provide benchmark optimum solutions for evaluating performance of the iterative method.

A common approach to characterize query behavior is to generate experimental queries by randomly selecting relations from a problem set. However, random selection of relations is not an accurate model of the access patterns occurring in real-world database systems. As the number of queries increases, random selection tends to create roughly equal joint-access probabilities between all relation pairs; usage patterns of this type create little incentive to allocate relations in separate clusters. In realistic systems, access patterns are not random -- some relations and relation sets have higher joint-access probabilities than others. These differences in access patterns create incentives to form clusters of relations, allocated at different sites.

To create more realistic data access patterns, our experimental design includes the concept of application data sets. Each experimental setup assumes a specified number of applications. Each application processes queries against a defined set of relations. Each query randomly selects relations from the application's set of relations, rather than from all available relations in the system. We assume each application is executed from a single site. If no overlap exists between application data sets, distributed allocation simply requires placing each relation at the site of the application. However, most often overlap exists between the data sets of different applications. This overlap in usage creates interesting distributed data design problems.

We model the overlap between the data sets used by different applications by a distribution function based on Zipf's Law [Knuth 1973]. Zipf's Law can be stated as:

$p_1 = c/1, p_2 = c/2, ..., p_N = c/N$,
where $c = 1/H_N$, and H_N is the Nth harmonic number.

We use an approximation to Zipf's law provided in [Knuth 1973] (page 398, equation 13):

$p_1 = c/1^{1-\theta}, p_2 = c/2^{1-\theta}, ..., p_N = c/N^{1-\theta}$, where $c = 1/H_N^{(1-\theta)}$, and $H_N^{(s)}$ is the Nth harmonic number of order s (i.e., $1^{-s} + 2^{-s} + ... + N^{-s}$).

Where p_i represents the probability that a given relation appears in i applications. (The procedure for assigning relations to applications is detailed in [Blankinship 1991].) When $\theta = 0$, this distribution

matches Zipf's Law. As θ increases, overlap between application data sets increases until all applications share the same set of relations; as θ decreases, overlap decreases until each application has a distinct data set with no relations shared between applications.

We present experimental results for the iterative design method we have proposed. Identical problem sets are used under the minimization of total time and the minimization of response time. The algorithms found in Section 3 are used for parametric studies. The dependent variable in the following experiments is the system cost of the required queries on the derived data allocation. We compare five levels of design optimization:

1. MFA - System cost of the Most Frequent Access data allocation used as a one-pass solution.
2. Apers - System cost of Apers' static data allocation used as a one-pass solution.
3. 1st Local Opt. - System cost of the first local optimal data allocation from the Iterative Heuristic algorithm.
4. Conv. Search - System cost of the final data allocation solution from the Search from Convergence algorithm.
5. Optimum - System cost of the global optimal data allocation obtained by exhaustive search. (Because of the complexity of the combined query optimization/data allocation problem, we find optimal solutions for only small problems.)

Because of the wide range of system costs found in the randomly generated experiments, we use the Apers method as a baseline for total time experiments, and the simple MFA method as a baseline for response time experiments. Each graph shows the baseline method fixed at 100% and records other cost lines as a percentage of the baseline method. In all graphs, each data point represents the average system cost for 100 randomly generated problems.

We study the effect of the following parameters as independent variables:

- Number of applications/sites in the problem.
- Number of relations per application.
- Relations per query, specified as the mean of a normal distribution with standard deviation 1.
- The amount of overlap between application data sets, θ.
- Number of queries, uniformly distributed over all applications/sites.

4.1 Total Time Experiments

This section evaluates the performance of the iterative design method to minimize total transmission time in the simple query environment.

Effect of Relations per Query

Figures 8 and 9 show performance of the iterative method and Apers method, as the number of relations per query is varied. Fixed parameter values are at the top of each graph. Figure 9 is for small problems which are tractable for exhaustive search. Optimum solutions are shown as benchmarks for performance of the iterative method.

The iterative method demonstrates increasing optimization benefits over the MFA method as the mean number of relations per query increases. The MFA method ignores transmissions between the relations in a query, which become increasingly important as the number of relations per query increases. The performance improvements of the iterative method over Apers method are largest in the middle ranges of the graphs; with a very small number of relations, little optimization potential exists; a similar situation occurs with a large number of relations in a simple query because all relations tend to cluster at a single site.

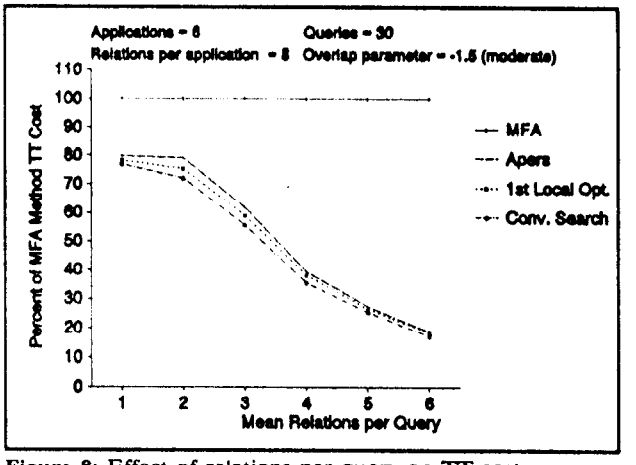

Figure 8: Effect of relations per query on TT cost.

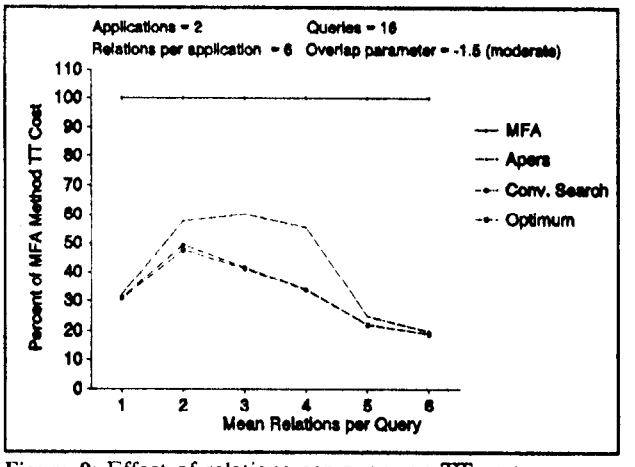

Figure 9: Effect of relations per query on TT cost.

Summary of Total Time Experiments

In addition to varying the number of relations per query, we generated problems while varying the number of applications, number of relations per application, and application overlap parameter. A

total of 4,400 problems were solved; 1,100 problems were small enough to be tractable by exhaustive search, while 3,300 were larger, more realistic problems:

- The iterative method always found solutions having equal or lower cost than Apers' method.
- The iterative method averaged within 2% of optimal on the 1,100 smaller problems solved for optimal solutions by exhaustive search.
- On the 3,300 larger problems, total transmission time was reduced compared to Apers' method on 1,374 (42%) of the problems; these reductions averaged 12% and ranged up to 96%.

4.2 Response Time Experiments

We perform a similar set of experiments minimizing response time in the simple query environment. While the change in cost objective is irrelevant to the MFA data allocation solution, Apers' static algorithm is predicated on total time minimization. Therefore, we use the lower cost data allocation from MFA or Apers' algorithms as the starting one-pass solution. Since we know of no data allocation algorithms designed specifically for response time minimization, system designers currently must use total time algorithms. Thus, to evaluate this approach, we include Apers algorithm in our response time graphs. Our experimental data show that the iterative design method consistently finds data allocation solutions with lower system response time cost than any known method.

Effect of Number of Applications

Figure 10 demonstrates a consistent improvement of data allocation cost, with the final iterative method solution yielding 60% cost reductions over the MFA solution. The number of applications has little effect on the range of cost improvement.

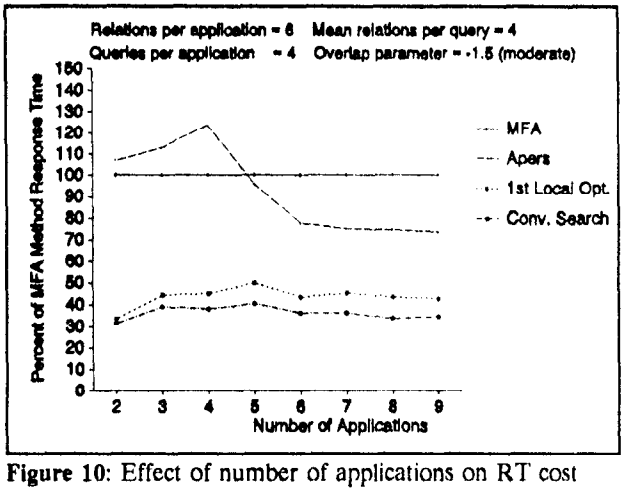

Figure 10: Effect of number of applications on RT cost

Effect of Number of Relations per Application

Figure 11 also shows relatively consistent response time improvements by the iterative method as the number of relations per application is varied; average response time improvement over MFA is 60% to 70%.

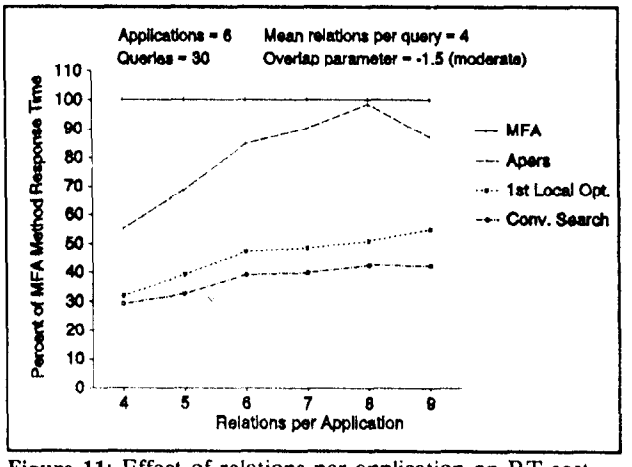

Figure 11: Effect of relations per application on RT cost

Effect of Relations per Query

When minimizing total time, we observed increasing optimization benefits over the MFA method as the mean number of relations per query increased. Figure 12 demonstrates the same effect when minimizing response time of queries. Apers method shows erratic response time performance when the mean number of relations per query is high.

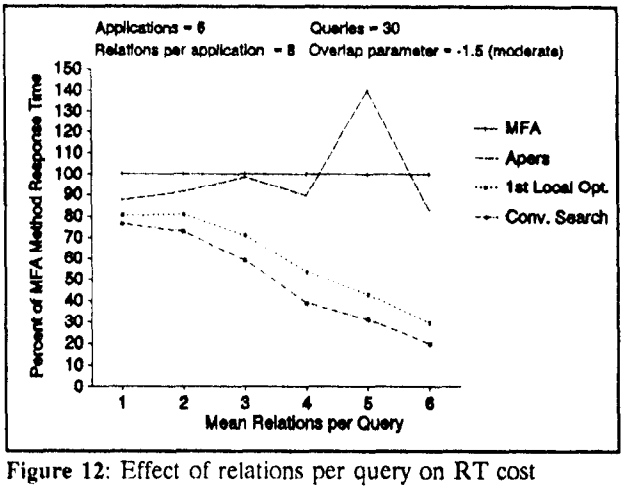

Figure 12: Effect of relations per query on RT cost

Effect of Application Overlap

Figure 13 shows a stable relationship between the optimization methods as we vary the application overlap parameter, theta. The iterative method reduces response time about 60% compared to MFA and about 50% compared to Apers method.

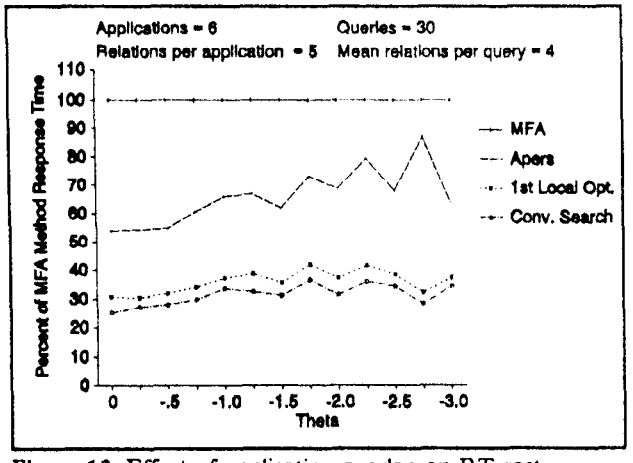

Figure 13: Effect of application overlap on RT cost

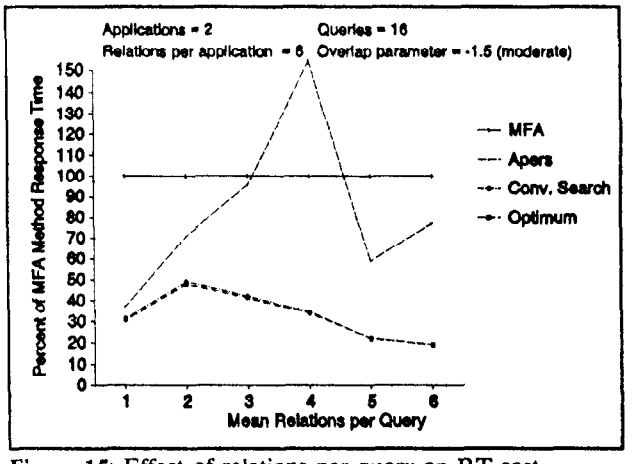

Figure 15: Effect of relations per query on RT cost

Comparison with Optimum Solutions Found by Exhaustive Search

As in the total time tests, we compare performance of the iterative method to optimal data allocation solutions found by exhaustive search. The graphs below show tests on small problems, which are tractable for exhaustive search. Figure 14 test the same independent variable (relations per application) as Figure 11, while Figure 15 displays results for the same variable (relations per query) as Figure 12.

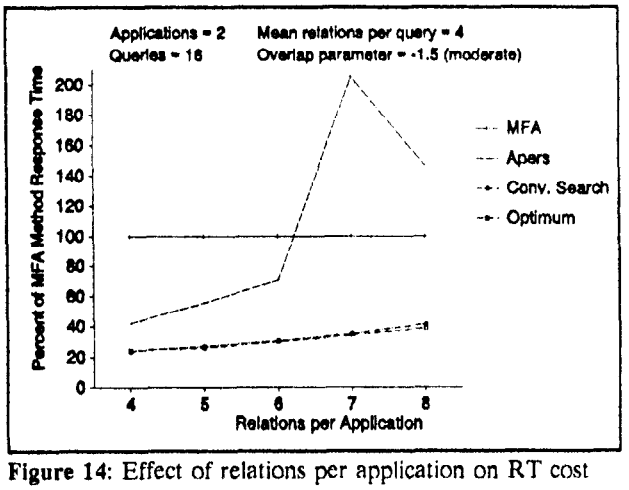

Figure 14: Effect of relations per application on RT cost

Summary of Response Time Experiments

On small test cases where optimal solutions are found by exhaustive search, solutions from the iterative method average 3% from optimum. An improved response time data allocation algorithm within the iterative framework, instead of the modified Apers' total time algorithm, would probably improve final solutions. Overall performance of the iterative method on these response time problems is significantly better than known alternatives.

4.3 Algorithm Efficiency

The table below demonstrates that the overall iterative design method is reasonably efficient. The number of calls to re-optimize all queries and actual run times for the heuristic are shown. Exhaustive search for the combined data allocation and query optimization problem can be performed by re-optimizing queries for each possible data allocation (i.e., s^r calls). Each line in the table represents an average for 5 problems. Run times were measured on an IBM PS/2 Model 80, with a 16Mhz 80386 processor, 80387 math co-processor, and 6MB of memory, running OS/2 1.2, with algorithms written in MicroSoft C 6.00. Test problems were generated with 6 relations per application and application overlap parameter = -1.5. These benchmarks are for minimizing total transmission time.

Table 1: Algorithm Efficiency

Number of			Iterative Method		Exhaustive Search	
Sites	Rel.	Queries	Iterations	Run Time	Search Space	Run Time
2	10	8	12	0.4 sec	1,024	9 sec
3	15	12	19	1.4 sec	14,348,907	*
4	18	16	37	3.8 sec	6.8×10^{10}	*
5	25	20	39	7.0 sec	3.0×10^{17}	*
6	29	24	68	21.6 sec	3.7×10^{22}	*
7	34	28	51	21.8 sec	5.4×10^{28}	*
8	36	32	58	30.4 sec	3.2×10^{32}	*
9	43	36	80	59.6 sec	1.1×10^{41}	*
10	52	40	96	111.0 sec	1.0×10^{52}	*

* - Runs not executed.

Since the iterative method for database design is applied at system design time and can be run off-line, performance efficiency is not a major concern.

5. Conclusions and Future Research Directions

This research is an important contribution to the understanding of the design tradeoffs between query

optimization and data allocation for distributed database design. We have developed and tested an iterative solution method as an efficient heuristic for simultaneously solving these two NP-hard problems. The experimental studies demonstrate that integrated solutions for the combined problem are often significantly below costs for one-pass methods. For both total time and response time cost objectives in the simple query environment, solutions are within 9% of optimum. Existing research is also extended by the development of a new response time cost model for the combined data allocation and query optimization problem. The iterative method is not only effective, but is also very flexible. It can accommodate most current query optimization and data allocation algorithms developed for distributed systems.

Planned extensions of this research include the application of our simple query environment insights on the iterative method to a general query environment. This research will include the development of a new response time data allocation algorithm for use within the iterative method. Finally, we plan to integrate the design problem of data redundancy into the iterative design method [Muro et al. 1985]. The number, placement, and effective use of data copies is an important design problem that is clearly interdependent with query optimization and data allocation.

References

[Apers 1982] Apers, P., "Query Processing and Data Allocation in Distributed Database Systems," Ph.D. Thesis, Vrije Universitit te Amsterdam, 1982.

[Apers 1988] Apers, P., "Data Allocation in Distributed Database Systems," ACM Transactions on **Database Systems**, Vol. 13, No. 3, September 1988, pp. 263-304.

[Blankinship 1991] Blankinship, R., "Query Optimization and Data Allocation on Distributed Database Systems: An Integrated Solution Approach," Ph.D. Thesis (in progress), 1991.

[Dowdy and Foster 1982] Dowdy, L. and D. Foster "Comparative Models of the File Assignment Problem," **ACM Computing Surveys**, Vol. 14, No. 2, June 1982, pp. 287-313.

[Eswaran 1974] Eswaran, K., "Placement of Records in a File and File Allocation in a Computer Network," **Proceedings of the 1974 IFIPS Conference**, 1974, pp. 304-307.

[Gavish and Segev 1986] Gavish, B. and A. Segev, "Set Query Optimization in Distributed Database Systems," **ACM Transactions on Database Systems**, Vol. 11, No. 3, Sept. 1986, pp. 265-293.

[Hevner and Yao 1979] Hevner, A. and S. Yao, "Query Processing in Distributed Database Systems", IEEE **Transactions on Software Engineering**, Vol. SE-5, May 1979, pp. 177-187.

[Hevner and Rao 1988] Hevner, A. and A. Rao, "Distributed Data Allocation Strategies," Chapter 3 in **Advances in Computers**, Vol. 27, Academic Press, Inc., pp. 121-155.

[Hevner and Yao 1987] Hevner, A. and S. Yao, "Querying Distributed Databases on Local Area Networks," **Proceedings of the IEEE**, Vol. 75, No. 5, May 1987, pp. 563-572.

[Knuth 1973] Knuth, D., **The Art of Computer Programming, Volume 3, Sorting and Searching**, Addison-Wesley, Inc., 1973.

[Loomis and Popek 1976] Loomis, M. and G. Popek, "A Model for Data Base Distribution", in "Trends and Applications: Computer Networks", IEEE Computer Society, pp. 162-169.

[Mukkamala et al. 1988] Mukkamala, R., S. Bruell, and R. Shultz, "Design of Partially Replicated Distributed Database Systems: An Integrated Methodology", **Proceedings of the 1988 ACM SIGMETRICS Conference**, Santa Fe, 1988, pp. 187-196.

[Muro et al. 1985] Muro, S., T. Ibaraki, M. Hidehiro, and T. Hasegawa, "Evaluation of the File Redundancy in Distributed Database Systems," **IEEE Transactions on Software Engineering**, Vol. SE-11, No. 2, Feb. 1985, pp. 199-204,

[Ozsu and Valduriez 1991] Ozsu, M. and P. Valduriez, **Principles of Distributed Database Systems**, Prentice-Hall, Inc., 1991.

[Sacca and Wiederhold 1985] Sacca, D. and G. Wiederhold, "Database Partitioning in a Cluster of Processors," **ACM Transactions on Database Systems**, Vol. 10, No. 1, 1985, pp. 29-56.

[Wah and Lien 1985] Wah, B. and Y. Lien, "Design of Distributed Databases on Local Computer Systems with a Multiaccess Network," **IEEE Transactions on Software Engineering**, Vol. SE-11, No. 7, July 1985, pp. 606-619.

[Yu and Chang 1984] Yu, C. and C. Chang, "Distributed Query Processing," **ACM Computing Surveys**, Vol. 16, No. 4, December 1984, pp. 399-433.