12-31-1995

Designing Distributed Database Systems for Efficient Operation

Sangkyu Rho
Seoul National University

Salvatore March
University of Minnesota

DESIGNING DISTRIBUTED DATABASE SYSTEMS FOR EFFICIENT OPERATION

Sangkyu Rho
Seoul National University

Salvatore T. March
Carlson School of Management
University of Minnesota

Abstract

Distributed database systems can yield significant cost and performance advantages over centralized systems for geographically distributed organizations. The efficiency of a distributed database depends primarily on the data allocation (data replication and placement) and the operating strategies (where and how retrieval and update query processing operations are performed). We develop a distributed database design approach that comprehensively treats data allocation and operating strategies, explicitly modeling their interdependencies for both retrieval and update processing. We demonstrate that data replication, join node selection, and data reduction by semijoin are important design and operating decisions that have significant impact on both the cost and response time of a distributed database system.

1. INTRODUCTION

Geographically distributed organizations must efficiently support local operations and must share information across the organization. With the emergence of commercial distributed database management systems (Ricciuti 1993; Richter 1994; The 1994), distributed database systems are becoming more common. Distributed database systems provide users with access to corporate databases that are maintained at different locations. Such systems can yield significant cost and performance advantages over centralized systems for geographically distributed organizations. These advantages include improved system performance, reduced system costs, and improved data availability (Ozsu and Valduriez 1991a, 1991b).

Given a computer network consisting of nodes (computers with processing and storage capabilities) connected by links (with data transmission speeds and capacities), judicious placement of data and processing capabilities can result in extremely efficient and responsive systems. However, inappropriate replication or placement of data or poor utilization of processing capabilities can result in high cost and poor system performance (Ozsu and Valduriez 1991b).

There are two aspects to distributed database design, data allocation and operating strategies. *Data allocation* includes the determination of units of data to allocate (termed file *fragments*) and the placement of copies of those units to nodes in the network. To enhance retrieval efficiency, the same data can be redundantly allocated to multiple nodes (i.e., data can be replicated). Of course, such redundancy increases update costs.

Operating strategies include operation allocation (or query optimization) and concurrency control strategies. *Operation allocation* defines where, how, and when retrieval and processing operations are performed (Yu and Chang 1984). Retrieval operations must be performed at a node containing the required data. Processing operations can be performed at any node; however, if the data is not located at the processing node, it must be sent there over the communication network. The order in which operations are performed can have a significant impact on performance. To reduce the amount of processing required, select and project operations are always performed before join operations. However, the order in which join operations are performed and the use of data reduction strategies must be determined.

The *concurrency control mechanism* is responsible for insuring that update operations are performed correctly and consistently, particularly when there are multiple copies of the data (Bernstein and Goodman 1981). Update operations must eventually be done at all nodes containing a copy of the affected data.

Data allocation and operation allocation are interdependent problems (Apers 1988). The optimal set of file fragments and their optimal allocation depend on how queries are processed (i.e., the operation allocation). However, the optimal operation allocation depends on where file fragments are located (i.e., the data allocation). Early work in this area focused either on data allocation, assuming either that there was no data redundancy or that a data copy would be pre-selected for each query, or on operation allocation (query optimization), assuming a given data allocation. Much of this work ignored the effects of update and concurrency control mechanisms.

We combine both data allocation and operation allocation in a single model. Our model includes data replication, update operations, a concurrency control mechanism, data reduction, join node selection, and join ordering. It evaluates both operating cost and response time. No current distributed database design approach includes all of these components.

We develop a genetic algorithm-based solution procedure and apply it to two example problems. The algorithm selects efficient data and operation allocations based either on minimum operating cost or on minimum response time criteria. In the example problems, we demonstrate that both operating cost and response time can be significantly reduced when replication, join node selection, and data reduction are considered, concluding that these are important components of a distributed approach.

The remainder of the paper is organized as follows: in the next section, we present a basic background in distributed database systems focusing on data allocation and operation allocation. In the following section, we briefly review the prior research. We then present our model and solution algorithm. Finally, we solve the example problems and compare our results in terms of total cost and response time with those obtained by other, more limited approaches.

2. DISTRIBUTED DATABASE SYSTEMS

In a distributed database system, data from a single conceptual database are maintained at various nodes in a computer network. The process of allocating data to nodes is termed *distribution design* or *data allocation* (Ceri, Pernici and Wiederhold 1987; Ozsu and Valduriez 1991a). Given a data allocation, user retrieval and update queries must be processed. Queries arise at some node and may update or retrieve data stored at any node. The process of determining how, when, and where queries are processed is termed, *query optimization* or *operation allocation*. The *concurrency control* mechanism specifies update processing constraints.

Typically, the data allocation and concurrency control strategy are determined at design time and change infrequently, if at all (there are research efforts in data migration strategies [Gavish and Sheng 1990]; however, this aspect of distributed system operation is beyond the scope of this paper). Operation allocation is typically done by a query optimizer within the distributed database management system either at compile time (e.g., R* [Lohman et al., 1985]) or at run time (e.g., Distributed INGRES [Epstein, Stonebraker and Wong 1978]). We argue that it is important to generate an efficient operation allocation for each known query at design time. This enables designers to estimate the system load and, perhaps, to pre-compile query execution strategies. Globally optimized query processing strategies may, in fact, be more efficient than one-at-a-time query optimizers.

The criteria against which data allocation and operation allocation decisions are evaluated typically relate to system costs and response time. System costs include data storage, network communication, and local processing. The determination of these as variable costs is itself a difficult problem depending on such factors as hardware utilization, the actual variable costs of operation (electricity, personnel), and the recovery of investment (initial cost, interest, depreciation, etc.).

Response time is the expected time that a data request spends in the system. It includes both processing time and the delays experienced in local data processing and data transmission. Response time is typically estimated using an open queueing network assuming Poisson arrival processes and exponential service time distributions (Kleinrock 1975; Cornell and Yu 1989).

For illustrative purposes, consider a bank having a headquarters and three regional offices. Suppose further that each has a computer system in a fully connected network. Each computer is described by CPU and disk capacities and unit costs. Each link in the network is described by speed, capacity, and unit transfer cost. Suppose that the database schema has three tables, Customer, Account, and Transaction (Figure 1). Each customer has some number of accounts against which transactions (deposits and withdrawals) are made. Each customer has a preferred regional office at which the customer does most of his/her banking (i.e., the office at which the accounts were opened). Of course, customers can go to any regional office. Each regional office must be able to process transactions for any customer (although they primarily process transactions for their own customers). Furthermore, regional offices and headquarters require access to data about various different customers, accounts, and transactions.

Figure 2 shows an example set of retrieval and update queries. Each is executed from each location with some selection criteria and some frequency. For example, Query R1 could be executed from headquarters once per day, selecting region 1 accounts (i.e., br-id = "Region 1"). It could be executed once per month from region 2 selecting region 2 accounts, and so forth. A distributed database system should allocate data and operations for efficient execution of known queries.

Customer (10,000 instances, 960,000 characters)

cd-id	Text	5
c-name	Text	20
ssn	Text	9
c-address	Text	30
c-city	Text	20
c-state	Text	2
c-zip	Text	10

Account (15,000 instances, 1,350,000 characters)

acc-no	Text	8
c-id	Text	5
br-id	Text	5
a-type	Text	2
a-status	Text	2
s-balance	Numeric	15.2
s-date	Date	8
c-balance	Numeric	15.2
period-interest	Numeric	15.2
ytd-interest	Numeric	15.2

Transaction (300,000 instances, 2,350,000 characters)

t-id	Text	10
acc-no	Text	8
loc-id	Text	5
date	Date	8
time	Time	8
amount	Numeric	15.2
t-type	Text	2
t-status	Text	2
t-ref	Text	20

Figure 1. Tables for an Example Distributed Database System

2.1 Data Allocation

Data allocation produces a subschema for each node of the distributed database system (Ceri, Pernici and Wiederhold 1987). Prior to data allocation, the units of data to allocate must be determined. This process is termed *fragmentation* (Navathe et al. 1984). There are two types of fragmentations: horizontal and vertical. Horizontal fragmentation groups records of a file that satisfy a selection condition (Ozsu and Valduriez 1991b). Vertical fragmentation groups attributes of a file that have a high probability of being accessed together (March 1983; Navathe et al. 1984).

Fragments must then be allocated to nodes (Dowdy and Foster 1982). Fragment allocation can be done either with or without replication.

Operating costs and response time of retrieval requests can be reduced by replication. Redundantly allocating a copy of each fragment to each node that references it allows all retrieval queries to be processed locally. However, such data replication increases the operating cost and response time of update queries as all copies of the referenced fragment must be updated. The exact effect of replication on update costs is dependent upon the concurrency control mechanism (Ram and Marsten 1991; Ram and Narasimhan 1990, 1994), a component of the distributed database operating strategy.

2.2 Operating Strategies

As discussed above, there are two components to operating strategies: operation allocation and concurrency control mechanism.

Operation allocation (or distributed query processing) involves three phases (Yu and Chang 1984): copy identification, reduction, and assembly.

In *copy identification* (also termed *materialization*), one or more copies of each fragment referenced by the query are selected for processing. The identification of appropriate fragment copies can play a critical role in determining the overall query processing costs (Martin, Lam and Russell 1990; Yu and Chang 1983, 1984).

Reduction applies only to join queries where the fragments to be joined are stored at different nodes. In it, *semijoins* are used to reduce the amount of data that must be transferred to accomplish the necessary joins. In a semijoin, denoted *reducer* → *reducee*, the unique join attribute values from the *reducer* fragment are transmitted to the node containing the *reducee* fragment. A record from the reducee is selected only if its join attribute matches one of the transmitted join values. Only the selected records are transmitted to the node containing the reducer and the join is performed there.

If all records in the reducee are selected, then no data reduction is achieved; the entire second fragment is sent to the node of the reducer. In this case, the semijoin strategy would not be *beneficial*. That is, it would not reduce the amount of data transferred to accomplish the join. Much of the work in distributed query optimization is devoted to identifying situations where semijoins are beneficial (e.g., Apers, Hevner and Yao 1983; Bernstein and Chiu 1981; Hevner and Yao, 1979).

In *assembly*, data are sent to the result node (if they are not already there) and final processing is performed (e.g., sorting and aggregations). Prior research typically assumes that reduced files are transmitted to the result node where *all* joins are performed. Furthermore, prior research typically ignores local processing costs, thus removing any consideration of join order. However,

a. Retrieval Queries

R1. Customer Statements

```
SELECT   c-id, c-name, c-address, c-city, c-state, c-zip, acc-no, s-balance,
         c-balance, period-interest, ytd-interest, t-id, t-type, amount.
FROM     Customer, Account, Transaction
WHERE    Customer.c-id = Account.c-id
AND      Account.acc-id = Transaction.acc-id
AND      Account.br-id = [region]
```

R2. Balance Inquiry

```
SELECT   c-id, c-name, acc-no, c-balance
FROM     Customer, Account
WHERE    Customer.c-id = Account.c-id
```

R3. Branch Status Report

```
SELECT   br-id, acc-no, c-balance
FROM     Account
WHERE    br-id = [region]
AND      acc-no = [specified]
```

b. Update Queries

U1. Adjust Account balance

```
UPDATE   Account
SET      c-balance = [new balance]
WHERE    acc-no = [specified]
```

U2. Maintain Customer Data

```
UPDATE   Customer
SET      c-address = [specified], c-city = [specified], c-state = [specified],
         c-zip = [specified]
WHERE    c-id = [specified]
```

U3. Record Transaction

```
INSERT INTO   Transaction
VALUES        ('t-id', ...... , 't-ref')
```

Figure 2. Queries for Example Database System

the nodes at which joins are performed and the join order (Mishra and Eich 1992) can significantly affect the overall query processing cost and response time. This research integrates join node selection and join order in a comprehensive model of data and operation allocation.

Concurrency control mechanisms specify how update processing is performed. In particular, they insure that replicated data are kept consistent. A number of distributed concurrency control mechanisms have been proposed (e.g., two-phase locking [Mohan, Lindsay and Obermarck 1986], timestamp-based [Bernstein, Shipman and Rothnie 1980], optimistic [Ceri and Owicki 1982]). Two-phase locking (2PL) is the most commonly implemented. Distributed 2PL, one variation of 2PL, is modeled in this research.

The next section presents a brief overview of prior research in distributed database design and distributed query optimization.

3. PRIOR RESEARCH

Several researchers have developed models for the combined data and operation allocation problem (Apers 1988; Blankinship, Hevner and Yao 1991; Cornell and Yu 1989). They do not, however, consider the effects of data replication or update queries and the requisite concurrency control mechanisms. Nor do they consider the effects of semijoins or join order. Noting that files are an inappropriate unit of allocation, Apers developed an approach to identifying file fragments for allocation. Blankinship, Hevner and Yao note that data and operation allocations should be evaluated by both cost and response time criteria.

Cornell and Yu develop a model that first decomposes queries into steps and then allocates files and query steps to nodes. Their cost model is simplistic, including only communication costs; however, their constraints are comprehensive, including local node data storage, I/O, and processing and communication capacity. In addition, their response time analysis includes queueing delays. Again, however, they do not consider the effects of data replication or update.

Ram and Narasimhan (1990) formulate a model which include data replication and a concurrency control mechanism, centralized two-phase locking (2PL) with a single, shared network directory. Their model includes queueing delays in local message processing operations. They assume that files are accessed independently and that, once accessed, files are sent from the storing node to the requesting node where all processing is done. In processing a complex distributed query, however, it may be more efficient to send files to intermediate nodes for processing before sending the results to the requesting node. Furthermore, local processing, except for messages, is ignored. Ram and Narasimhan (1994) formulate a similar model based on primary copy 2PL concurrency control mechanism.

March and Rho (1995) develop a model that also includes data replication, update queries, and a concurrency control mechanism. Their model treats both data and operation allocation in an integrated manner. Their cost model includes local node storage, I/O, and CPU processing costs as well as communication costs. Although operations can be allocated to any node, they assume that join order is predetermined. Furthermore, they do not include semijoins.

As discussed above, join order and the use of semijoins can have a significant impact on performance. Work in distributed query optimization dealing with semijoins (Apers, Hevner and Yao 1983; Bernstein and Chiu 1981; Hevner and Yao 1979; Yoo and Lafortune 1989) assumes that a data allocation is given.

In this paper, we extend the basic approach of March and Rho to incorporate a more comprehensive operation allocation model that includes reduction by semijoins and the determination of join order. We adopt Apers approach to identifying file fragments for allocation and the query decomposition of Cornell and Yu. We model costs and queueing delays in communications and in local data processing including both retrieval and update queries. We include two evaluation criteria: minimization of total operating cost and minimization of average response time. In the next section, we present our model and its solution method.

4. A MODEL AND ALGORITHM FOR DATA AND OPERATION ALLOCATION

We assume that the global database schema is given, along with a set of queries for which performance is to be optimized. Following March and Rho and Apers, we first determine min-term fragments for allocation from the selection and projection criteria of queries. Based on the retrieval queries R1, R2, and R3 of Figure 2, for example, each relation in Figure 1 could be horizontally partitioned into three fragments, each containing the instances for one region (e.g., Customer into Customer 1, Customer 2, and Customer 3).

We then transform the queries into subqueries on the min-term fragments, which in turn are decomposed into query steps. For example, retrieval query R3 from Region 1 would simply retrieve the corresponding min-term fragment (i.e., Account 1); however, R3 from headquarters would require a union of all three min-term fragments (i.e., Account 1, Account 2, and Account 3). Therefore, R3 can be decomposed into three subqueries (i.e., R1.1, R1.2, and R1.3) based on the fragments required. Query steps include all messages that need to be sent as well as the actual retrieval and processing that must be performed. If data needed for a query is not stored at the requesting node, it must send messages to the node(s) from which it will be retrieved. As we assume a distributed 2PL concurrency control mechanism, update queries require the set of messages.

The task is then to

(1) allocate min-term fragments to nodes (data allocation with replication),

(2) allocate retrieval query steps to nodes (copy identification), and for each join query,

(3) identify beneficial semijoins,

(4) determine join order, and

(5) allocate join operations to nodes (join node selection)

to minimize either total operating cost or average response time.

Following Cornell and Yu and March and Rho, the following notation is used: a(k,m) and b(k,m) are defined as the file fragments referenced by step m of query k. For message and local selection and projection steps only one file is referenced, hence, for those steps b(k,m) is null. For combine-fragment steps (joins and unions), a(k,m) and b(k,m) are temporary files generated in previous selection and projection, semijoin or combine-fragment steps. L_i is defined as the size of file fragment i (in characters) and L^M as the size of a message. The size of each file fragment is calculated from the problem description parameters. The size of each temporary file is estimated from the selection and projection conditions, semijoin, and join operations that produce them (see, e.g., Gardy and Peuch 1989). node(k) is defined as the origination node of query k, node(k,m) as the node at which step m of query k is performed; and node(i) as the node from which fragment i is accessed. For join steps, node(k,m) represents join node selection decision. Similarly, for message steps of retrievals, node(a(k,m)) represents copy identification decisions. Finally, copy(i,t) represents fragment allocation and it is 1 if fragment i is stored at node t, otherwise it is 0.

4.1 Total Operating Cost Model

The first performance model is designed to minimize total operating cost including communication, disk I/O, CPU processing, and storage, i.e.,

$$\text{Min Cost} = \sum_k f(k) \sum_m (COM(k,m) + IO(k,m) + CPU(k,m)) + \sum_t STO(t)$$

Where f(k) is the frequency of execution of query k per unit time, COM(k,m), IO(k,m), and CPU(k,m) are the respective costs of communication, disk I/O and CPU processing time for step m of query k, and STO(t) is the cost of storage at node t per unit time. Expressions for these cost components are summarized in Appendix 1.

To be feasible, a data and operation allocation must not exceed system resource capacities. We consider communication link, disk I/O, CPU, and storage capacities as constraints. These are assumed to be given.

4.2 Average Response Time Model

The average response time of query k can be decomposed into three parts: communication ($R_{COM}(k)$), disk I/O ($R_{IO}(k)$), and CPU ($R_{CPU}(k)$). The objective, then, is to

$$\text{Min } R_T = \frac{\sum_k f(k)\ (R_{COM}(k) + R_{IO}(k) + R_{CPU}(k))}{\sum_k f(k)}$$

We assumed M/M/1 queueing models for communication links, disks, and CPU's. Expressions for the above response time components are summarized in Appendix 2.

4.3 A Genetic Algorithm Solution Procedure

One of the difficulties facing researchers in this area is tractability. Data and operation allocation are interrelated problems, each of which is NP-hard (Eswaran 1974; Hevner 1979). To address the tractability problem, we developed a genetic algorithm-based solution procedure (Goldberg 1989). A genetic algorithm was chosen for several reasons. First, genetic algorithms have been successfully applied to similar complex, combinatoric, real-world problems including distributed database design (March and Rho 1995). Second, genetic algorithms are robust in that they work well even in discontinuous, multimodal, noisy search spaces (Goldberg 1989). Genetic algorithm-based solution methods can easily incorporate very complex and nonlinear cost models such as ours. Third, genetic algorithms result not only in a "best" solution, but also in a pool of good solutions. The set of solutions in the final pool provides significant intuition into the effects of design alternatives. For example, if all solutions in the final pool store a given file at a particular node, the designer would be reasonably confident that it is important to store that file at that node.

Our distributed database design algorithm contains a genetic algorithm within a genetic algorithm (adapted from Rho and March [1994] and summarized in Appendix 3). The outer genetic algorithm addresses data allocation. The inner genetic algorithm addresses operation allocation. A nested approach is advantageous over a standard approach because it can more easily handle the dependency between data allocation and operation allocation. As discussed above, the feasibility of an operation allocation is dependent on the data allocation — each retrieval operation must be allocated to a node containing the required data. It is very difficult to enforce this type of constraint with a standard generic algorithm.

Furthermore, a nested approach allows us to easily incorporate different operation allocation models. Such flexibility is desirable in distributed database design, since different distributed database management systems utilize different query processing models (i.e., query optimizers). The genetic algorithm is written in C++ and runs in a UNIX environment. Its run time depends on problem size (i.e., the number of nodes and queries) and on algorithm parameters (poolsize and number of iterations).

In the remainder of this section, we briefly describe how a solution is represented in the genetic algorithm. Details of the algorithm are presented in Rho (1995). The solution representation for the outer genetic algorithm represents the fragment allocation. The solution representation for the inner genetic algorithm consists of four parts, each representing one of the four types of decisions in our operation allocation model: (1) copy identification, (2) beneficial semijoin identification, (3) join order, and (4) join node selection. Figure 3 shows the representation of one solution. Each part of the representation is discussed below.

a. Outer Algorithm Representation

Fragment	Fragment Allocation
Customer 1	1110
Account 1	1100
Transaction 1	0010
Customer 2	1100
Account 2	1110
Transaction 2	0100
Customer 3	1000
Account 3	0001
Transaction 3	1000

b. Inner Algorithm Representation

Query	Origination Node	Copy Id.	Semi-join	Join Order	Join Node
R1.1	HQ	2 0 2	01 10	1 2	0 0
R1.1	Region 1	2 1 2	01 10	1 2	1 1
R1.2	Region 2	1 2 1	01 10	1 2	2 2
R1.3	Region 3	0 3 0	02 20	2 3	3 3
R2.1	HQ	0 0	00		0
R2.1	Region 1	1 1	00		1
R2.1	Region 2	2 1	01		2
R2.2	HQ	0 0	00		0
R2.2	Region 1	1 1	00		1
R2.2	Region 2	1 2	01		2
R2.3	HQ	0 3	00		0
R2.3	Region 3	0 3	01		3
R3.1	HQ	0			
R3.1	Region 1	1			
R3.2	HQ	0			
R3.2	Region 2	2			
R3.3	Region 3	3			

Figure 3. An Example Solution Representation

The fragment allocation is represented as sets of n bits, one set for each fragment, where n is the number of nodes in the network. A bit has a value of 1 if the corresponding file fragment is allocated to the corresponding node. It has a value of 0 otherwise. The solution illustrated in Figure 3.a (1110 1100 0010 ... 1000) stores Customer 1 at Headquarters, Region 1, and Region 2; Account 1 at Headquarters and Region 1; Transaction 1 at Region 2; etc.

Copy identification decisions are represented by a vector with a position for each fragment referenced by a query. Each value in the vector is the node from which the fragment is accessed. Query R1.1 requires fragments Customer 1, Account 1, and Transaction 1 (see Figure 2). The copy identification illustrated in Figure 3.b for this query, originating at Headquarters (the vector (2 0 2) in the Copy Id column), specifies the use of Customer 1 from Region 2, Account 1 from Headquarters, and Transaction 1 from Region 2.

Semijoin decisions are represented as sets of 2 bits, one set for each join in the query. If a semijoin is to be performed, the value of the bit corresponding to the reducer file is set to 1, otherwise it is 0. Query R.1 requires two joins (i.e., three fragments must be joined). The semijoin decision for this query, originating at Headquarters (the bit sets (01 10) in the Semijoin column of Figure 3.b) specifies the use of semijoins Account 1→ Customer 1 and Account 1 → Transaction 1.

Join order decisions are represented as a list of joins, where the sequence indicates the order in which joins are performed. The join order decision for query R1.1 originating at Headquarters (the list (1 2) in the Join Order Column of Figure 3.b), specifies that the join between Customer 1 and Account 1 (i.e., the first join in the query) is performed before the join with Transaction 1 (the second join in the query).

Join node decisions are represented by a vector with a position for each join in the query. Each value in the vector is the node at which the join is performed. The join node decision for query R1.1 originating at Headquarters (i.e., the vector (0 0) in the Join Node column of Figure 3.b), specifies that both joins are performed at Headquarters.

5. COMPARISON OF MODELS

In this section, we compare our model with prior models in terms of total operating cost and average response time and demonstrate that data replication, join node selection, and data reduction by semijoin can have significant effects on both the operating cost and response time of a distributed database system.

In order to systematically compare our model with prior models, we classify distributed database design models based on their data and operation allocation strategies. We consider two types of data allocation strategies: No replication (NR) and Replication (R). We consider three types of operation allocation strategies. Operation Allocation Strategy 1 (OA1) is similar to that of Ram and Narasimhan (1991, 1994). It includes only copy identification. It ignores data reduction by semijoin and assumes that all joins are performed at the result node in a predetermined order. Operation Allocation Strategy 2 (OA2) includes join node selection as well as copy identification. Like OA1, however, it ignores reduction by semijoins and assumes that joins are performed in a predetermined order. It is the model used in Cornell and Yu and in March and Rho. Finally, Operation Allocation Strategy 3 (OA3) is the model presented in this paper. It integrates copy identification, join node selection, beneficial semijoin selection, and join ordering. This results in six different distributed database design models (NR-OA1, R-OA1, etc.; see Figure 4.). The distributed database design models of Ram and Narasimhan (1990, 1994), Cornell and Yu, and March and Rho correspond to R-OA1, NR-OA2, and R-OA2, respectively.

We solved the example problem both to minimize total operating costs (Figure 5) and to minimize average response time (Figure 6), for each model described above (see Table 1) using the genetic algorithm. Cost and response times are reported relative to the base case NR-OA1. The example problems were solved in under six hours on a Sun Sparc 20 workstation. The poolsize and number of iteration for the outer algorithm were 50 and 1,000, respectively; and those for the outer algorithm were 300 and 5,000, respectively.

In Figure 5, columns represent the total operating cost of the best (i.e., "minimum" total operating cost) solution found for each model. Dotted lines represent the average response time. In Figure 6, columns and dotted lines represent the minimum average response time and total operating cost, respectively.

As shown in Figure 5, replication reduced the minimum operating cost significantly across different operation allocation strategies. However, its effect becomes smaller as the operation allocation strategy becomes more comprehensive. Join node selection (OA2) reduced the cost significantly when replication was not allowed. However, it did not when replication was allowed. This is not unreasonable since the problem was retrieval intensive. In a retrieval intensive problem, it is likely that nodes become self-contained (i.e., each node contains all the data necessary to meet its retrieval requests, therefore no communication is required). When all or most of the nodes become self-contained, join node selection is not likely to have significant effects on the minimum cost. Semijoins and join order (OA3) reduced the minimum operating cost significantly. Further analysis of the solutions revealed that the effect was mainly due to semijoins. It is not surprising since two-join queries were the most complex query type in the problem. As expected, the average response time was reduced as the minimum total operating cost was reduced except for OA3. A 10% reduction in the minimum total operating cost was achieved at a slight increase in the average response time.

Operation Allocation Strategy	Data Allocation Strategy	
	No Replication (NR)	Replication (R)
Copy Identification (OA1)	NR-OA1 (Base Case)	R-OA1 (Ram and Narasimhan 1994)
+Join Node Selection (OA2)	NR-OA2 (Cornell and Yu 1989)	R-OA2 (March and Rho 1995
+Beneficial Semijoin & Join Order (OA3)	NR-OA3	R-OA3

Figure 4. Types of Distributed Database Design Models

Table 1. Minimum Cost by Data and Operation Allocation Strategies

Objective	Data Allocation Strategy	Operation Allocation Strategy		
		OA1	OA2	OA3
Min Total Operating Cost (Average Response Time)	No Replication	59499.5 (18.788)	46095.6 (17.854)	29533.5 (7.197)
	Replication	40882.0 (14.843)	40771.8 (14.760)	26597.8 (7.354)
Min Average Response Time (Total Operating Cost)	No Replication	17.878 (60941.4)	14.940 (48128.7)	7.201 (29832.8)
	Replication	14.525 (43156.1)	13.889 (44159.2)	6.851 (27609.0)

As shown in Figure 6, data and operation allocation strategies have similar effects on the minimum average response time. Replication reduced the minimum response time across different operation allocation strategies. However, the effect is not as significant as on the minimum operating cost. Join node selection (OA2) reduced the response time. Unlike the minimum operating cost criteria, join node selection slightly reduced the response time when replication was allowed. Semijoins and join order (OA3) reduced the minimum response time significantly. Often the total operating costs were reduced as the minimum average response time was reduced. This is not always the case, however, as illustrated in the solution for OA2 with replication where a reduction in the minimum average response time was accompanied by a slight increase in operating cost (due primarily to an increase in storage costs).

An update intensive variation of the example problem was also solved. These results are presented in Figures 7 and 8. As expected, replication was not as effective (since it can significantly increase update costs and response time). As illustrated in Figure 7, minimizing total operating costs resulted in a significant increase in the minimum average response time for OA2. Again, semijoins and join order (OA3) reduced both the operating costs and response time significantly.

Although limited in scope, the results demonstrate that replication, join node selection, and data reduction by semijoin can have significant impact on the operating cost and response time of a distributed database system. The results also suggests that there can be trade-offs between total operating costs and average response time.

6. SUMMARY AND FUTURE RESEARCH

We developed a comprehensive distributed database design model that treats data allocation and operating strategies in an integrated manner. Our model includes data replication, a concurrency control mechanism, data reduction by semijoin, join node selection, and join ordering, aspects of distributed database design that are typically treated in isolation in prior work. We

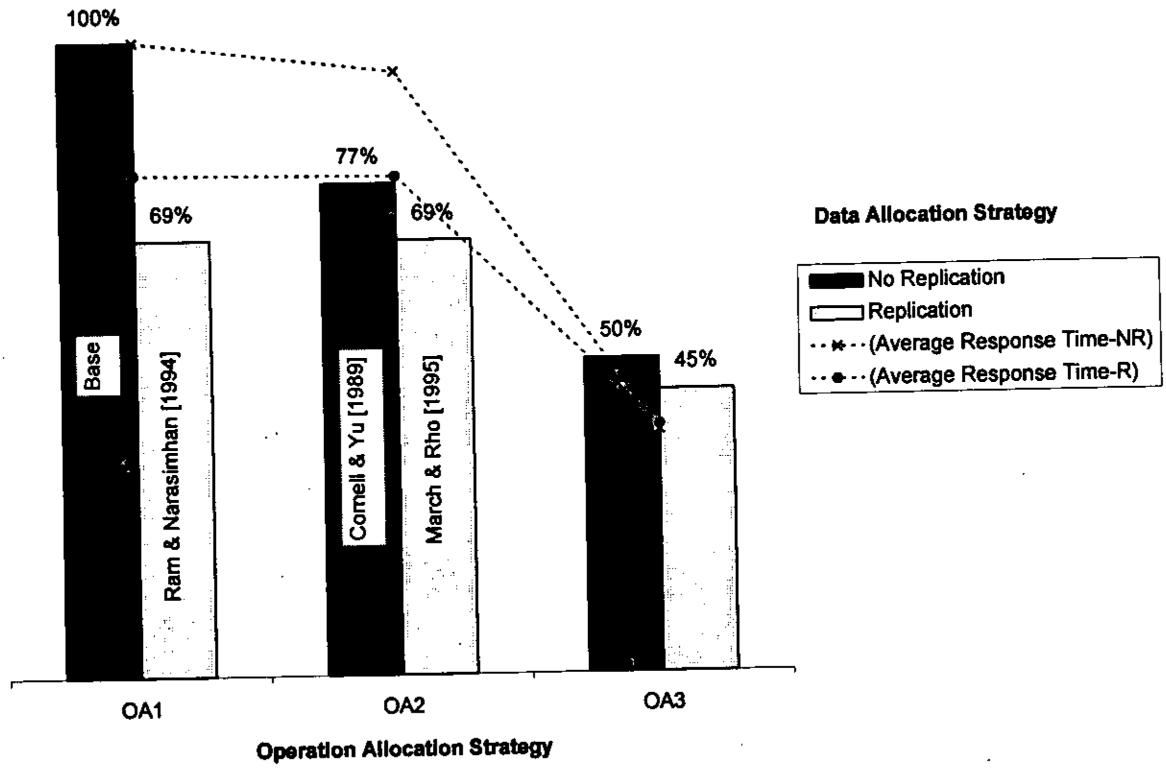

Figure 5. Effects of Data Operation Allocation Strategies on the Minimum Total Operating Cost

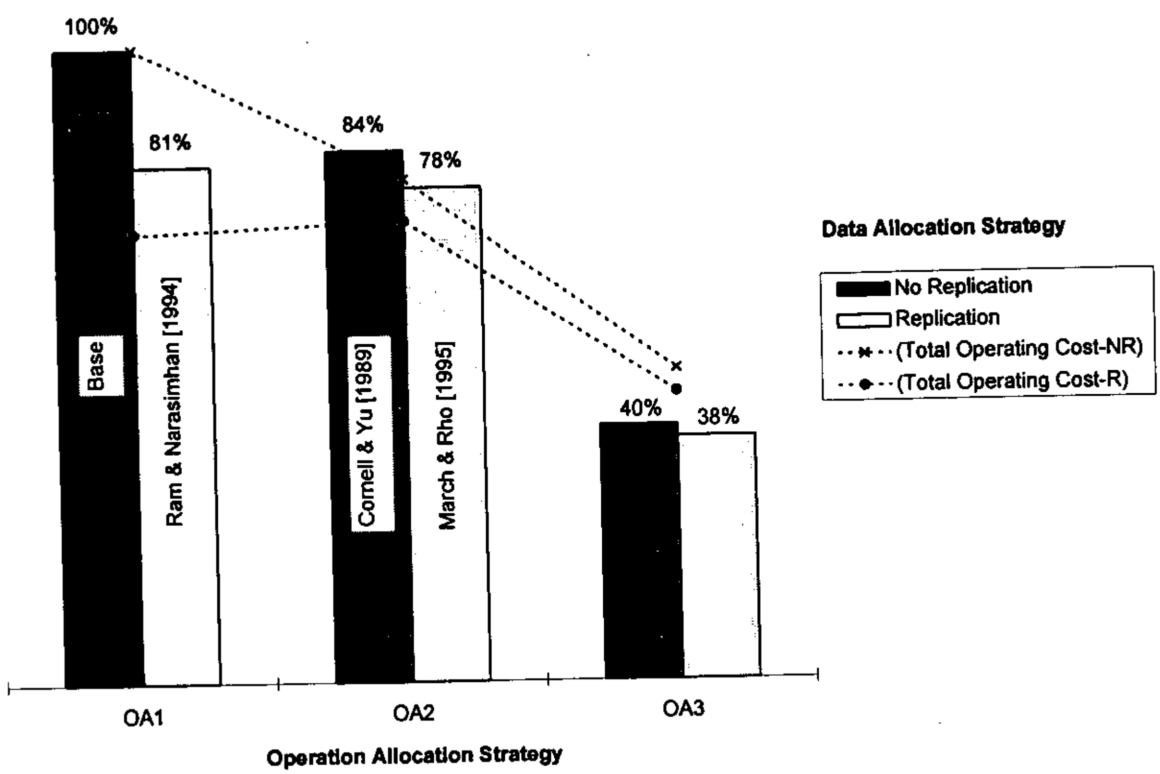

Figure 6. Effects of Data and Operation Allocation Strategies on the Minimum Average Response Time

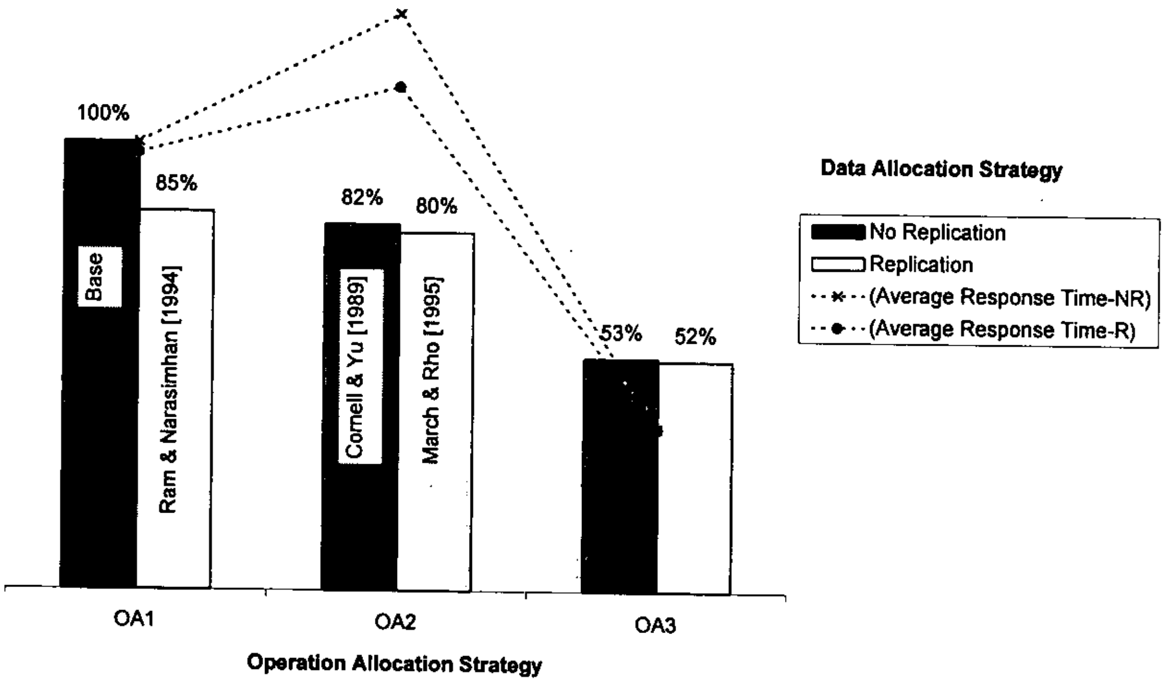

Figure 7. Effects of Data and Operation Allocation Strategies on the Minimum Total Operating Costs

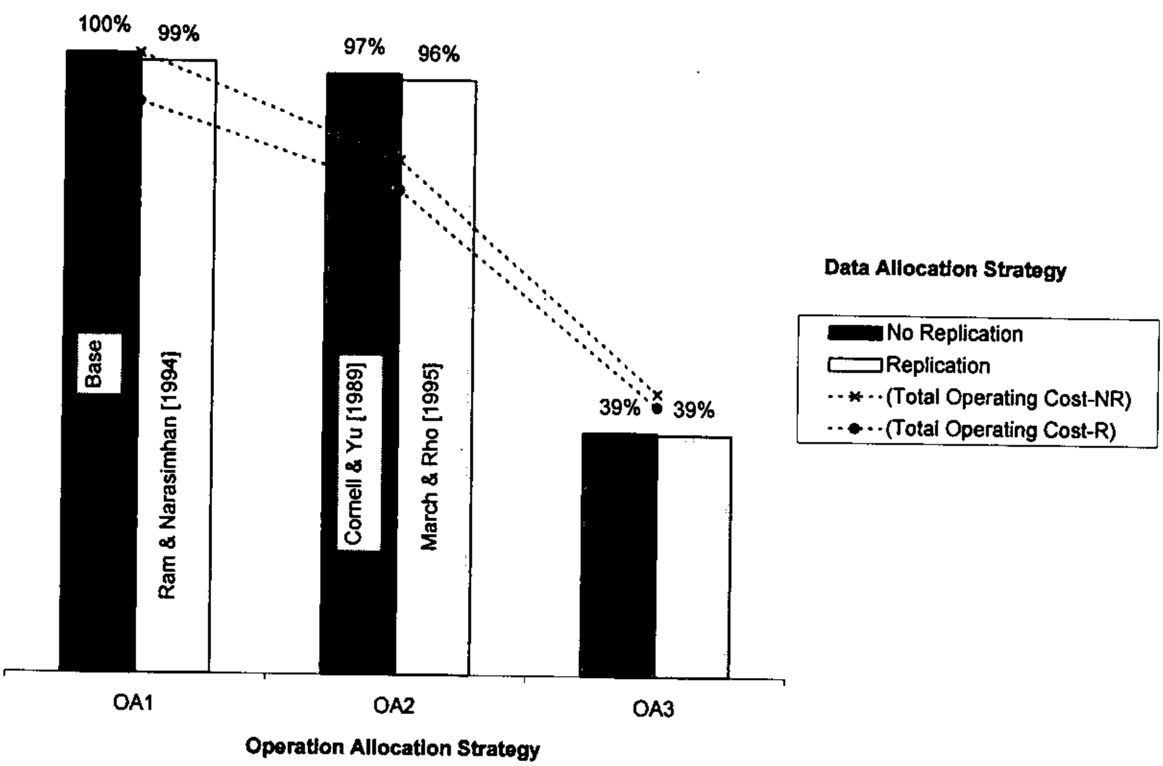

Figure 8. Effects of Data and Operation Allocation Strategies on the Minimum Average Response Time

developed a nested genetic algorithm to solve this problem formulation. Using example problems, we have demonstrated that replication, join node selection, and reduction by semijoin can each have significant impact on the efficiency of a distributed database system.

There are several areas for future research. First, additional experimentation must be done with a variety of problem types. We are currently performing such a study (Rho 1995). Second, the model must be evaluated and verified in a realistic environment. Selected solutions should be implemented and their performance measured in real organizational settings. Third, the effects of data and operation allocation strategies on the efficiency of distributed database systems should be further analyzed under various conditions (e.g., different types of networks with different performance parameters) using real business problems. Fourth, much work is needed to develop and compare alternative solution algorithms. Possible candidates include simulated annealing, partial enumeration techniques, and Lagrangian relaxation. Finally, the model itself can be extended to be more realistic. Possible extensions include the modeling of data availability, dynamic system loads, parallel data access and different processing priorities.

7. REFERENCES

Apers, P. M. G. "Data Allocation in Distributed Database Systems." *ACM Transactions on Database Systems*, Volume 13, Number 3, September 1988, pp. 263-304.

Apers, P. M. G.; Hevner, A. R.; and Yao, S. B. "Optimization Algorithms for Distributed Queries." *IEEE Transactions on Software Engineering*, Volume SE-9, Number 1, January 1983, pp. 57-68.

Bernstein, P. A., and Chiu, D. W. "Using Semi-Joins to Solve Relational Queries." *Journal of the ACM*, Volume 28, Number 1, January 1981, pp. 25-40.

Bernstein, P. A., and Goodman, N. "Concurrency Control in Distributed Database Systems." *ACM Computing Surveys*, Volume 13, Number 2, June 1981, pp. 185-222.

Bernstein, P. A.; Shipman, D. W.; and Rothnie, J. B. "Concurrency Control in a System for Distributed Databases (SDD-1)." *ACM Transactions on Database Systems*, Volume 5, Number 1, March 1980, pp. 18-51.

Blankinship, R.; Hevner, A. R.; and Yao, S. B. "An Iterative Method for Distributed Database Design." *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, Barcelona, Spain, September 1991, pp. 389-400.

Ceri, S., and Owicki, S. "On the Use of Optimistic Methods for Concurrency Control in Distributed Databases." *Proceedings of Sixth Berkeley Workshop on Distributed Data Management and Communication Networks*, Berkeley, California, February 1982, pp. 117-130.

Ceri, S.; Pernici, B.; and Wiederhold, G. "Distributed Database Design Methodologies." *Proceedings of the IEEE*, Volume 75, Number 5, May 1987, pp. 533-546.

Cornell, D. W., and Yu, P. S. "On Optimal Site Assignment for Relations in the Distributed Database Environment." *IEEE Transactions on Software Engineering*, Volume 15, Number 8, August 1989, pp. 1004-1009.

Dowdy, L. W., and Foster, D. V. "Comparative Models of the File Assignment Problem." *ACM Computing Surveys*, Volume 14, Number 2, June 1982, pp. 287-314.

Eswaran, K. P. "Placement of Records in a File and File Allocation in a Computer Network." *Information Processing '74*, Stockholm, 1974, pp. 304-307.

Epstein, R.; Stonebraker, M.; and Wong, E. "Query Processing in a Distributed Relational Database System." *Proceedings of ACM SIGMOD*, Austin, Texas, May 1978.

Gardy, D., and Puech, C. "On the Effects of Join Operations on Relation Sizes." *ACM Transactions on Database Systems*, Volume 14, Number 4, December 1989, pp. 574-603.

Gavish, B., and Sheng, O. R. L. "Dynamic File Migration in Distributed Computer Systems." *Communications of the ACM*, Volume 33, Number 2., February 1990, pp. 177-189.

Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Massachusetts: Addison-Wesley, 1989.

Hevner, A. R. *The Optimization of Query Processing on Distributed Database Systems*. Ph.D. Thesis, Purdue University, 1979.

Hevner, A. R., and Yao, S. B. "Query Processing in Distributed Database Systems." *IEEE Transactions on Software Engineering*, Volume SE-5, Number 3, May 1979, pp. 177-187.

Kleinrock, L. *Queueing Systems: Theory*. New York: John Wiley & Sons, 1975.

Lee, H., and Sheng, O. R. L. "A Multiple Criteria Model for the Allocation of Data Files in a Distributed Information Systems." *Computers and Operations Research*, Volume 21, 1992, pp. 21-33.

Lohman, G. M.; Mohan, C.; Haas, L. M.; Daniels, D.; Lindsay, B. G.; Selinger, P. G.; and Wilms, P. F. "Query Processing in R*." In W. Kim et al. (Editors), *Query Processing in Database Systems*. Berlin: Spring-Verlag, 1985, pp. 31-47.

March, S. T. "Techniques for Structuring Database Records." *ACM Computing Surveys*, Volume 15, Number 1, March 1983, pp. 45-79.

March, S. T., and Rho, S. "Allocating Data and Operations to Nodes in Distributed Database Design." *IEEE Transactions on Knowledge and Data Engineering*, Volume 7, Number 2, April 1995, pp. 305-317.

Martin, T. P.; Lam, K. H.; and Russell, J. I. "Evaluation of Site Selection Algorithms for Distributed Query Processing." *Computer Journal*, Volume 33, Number 1, February 1990, pp. 61-70.

Mishra, P., and Eich, M. H. "Join Processing in Relational Databases." *ACM Computing Surveys*, Volume 24, Number 1, March 1992, pp. 63-113.

Mohan, C.; Lindsay, B.; and Obermarck, R. "Transaction Management in the R* Distributed Database Management System." *ACM Transactions on Database Systems*, Volume 11, Number 4, December 1986, pp. 378-396.

Navathe, S.; Ceri, S.; Wiederhold, G.; and Dou, J. "Vertical Partitioning Algorithms for Database Design." *ACM Transactions on Database Systems*, Volume 9, Number 4, December 1984, pp. 680-710.

Ozsu, M., and Valduriez, P. "Distributed Database Systems: Where Are We Now?" *IEEE Computer*, August 1991a, pp. 68-78.

Ozsu, M., and Valduriez, P. *Principles of Distributed Database Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1991b.

Ram, S., and Marsten, R. E. "A Model for Database Allocation Incorporating a Concurrency Control Mechanism." *IEEE Transactions on Knowledge and Data Engineering*, Volume 3, 1991, pp. 389-395.

Ram, S., and Narasimhan, S. "Allocation of Databases in a Distributed Database System." In J. I. DeGross, M. Alavi, and H. Oppelland (Editors), *Proceedings of the Eleventh International Conference on Information Systems*, Copenhagen, Denmark, December 1990, pp. 215-230.

Ram, S., and Narasimhan, S. "Database Allocation in a Distributed Environment: Incorporating a Concurrency Control Mechanism and Queuing Costs." *Management Science*, Volume 40, Number 8, August 1994, pp. 969-983.

Rho, S. *Distributed Database Design: Allocation of Data and Operations to Nodes in Distributed Database Systems*. Unpublished Ph.D. Thesis, University of Minnesota, May 1995.

Rho, S., and March, S. T. "A Nested Genetic Algorithm for Distributed Database Design." *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, January 1994, pp. 33-42

Ricciuti, M. "DBMS Vendors Chase Sybase for Client/Server." *Datamation*, Volume 39, July 1, 1993, pp. 27-28.

Richter, J. "Distributing Data." *Byte*, June 1994, pp. 139-148.

The, L. "Distribute Data Without Choking the Net." *Datamation*, Volume 40, January 7, 1994, pp. 35-36.

Yoo, H., and Lafortune, S. "An Intelligent Search Method for Query Optimization by Semijoins." *IEEE Transactions on Knowledge and Data Engineering*, Volume 1, Number 2, June 1989, pp. 226-237.

Yu, C. T., and Chang, C. C. "Distributed Query Processing." *ACM Computing Surveys*, Volume 16, Number 4, December 1984, pp. 399-433.

Yu, C. T., and Chang, C. C. "On the Design of a Distributed Query Processing Algorithm." *Proceedings of the ACM-SIGMOD International Conference on the Management of Data*, San Jose, 1983, pp. 30-39.

Appendix 1. Total Operating Cost

$$COM(k,m) = \sum_{t} \sum_{p \neq t} H(k,m,t,p)\, c_{tp}$$

where c_{tp} is the communication cost per character from node t to p.

For message steps of retrievals,

$H(k,m,t,p) = L^M$ if $t = node(k)$ and $p = node(a(k,m))$
$H(k,m,t,p) = 0$ otherwise

where L^M is the size of a message, node(k) is the origination node of query k, node(i) is the node at which file fragment i is located.

For join steps,

$H(k,m,t,p) = L_{a(k,m)} + L_{b(k,m)}$	if $t = node(a(k,m)) = node(b(k,m))$ and $p = node(k,m)$
$H(k,m,t,p) = L_{a(k,m)}$	if $t = node(a(k,m))$ and $t \neq node(b(k,m))$ and $p = node(k, m)$
$H(k,m,t,p) = L_{b(k,m)}$	if $t \neq node(a(k,m))$ and $t = node(b(k,m))$ and $p = node(k, m)$
$H(k,m,t,p) = 0$	otherwise.

where L_i is the size of file fragment i (in characters), a(k,m) and b(k,m) are the file fragment referenced by step m of query k, and node(k,m) is the node at which step m of query k is processed.

For a final step,

$H(k,m,t,p) = L_{a(k,m)}$	if $t = node(a(k,m))$ and $p = node(k)$
$H(k,m,t,p) = 0$	otherwise.

For send-message steps of updates,

$H(k,m,t,p) = L^M$	if $t = node(k)$ and $copy(a(k,m), p) = 1$
$H(k,m,t,p) = 0$	otherwise

where copy(i,t) is 1 if fragment I is stored at node t, and 0 otherwise.

For receive-message steps of updates,

$H(k,m,t,p) = L^M$	if $copy(a(k,m), t) = 1$ and $p = node(k)$
$H(k,m,t,p) = 0$	otherwise.

$$IO(k, m) = \sum_{t} O(k,m,t)\, d_t$$

where d_t is the cost per disk I/O at node t.

For selection and projection steps,

$O(k,m,t) = D_{kmt}$ if $t = node(a(k,m))$
$O(k,m,t) = 0$ otherwise

where D_{kmt} is the number of disk I/Os required to process step m of query k at node t.

For join steps,

$O(k,m,t) = F_{a(k,m)t}$	if $t \neq node(k, m)$ and $t = node(a(k,m))$ and $t \neq node(b(k,m))$
$O(k,m,t) = F_{b(k,m)t}$	if $t \neq node(k, m)$ and $t \neq node(a(k,m))$ and $t = node(b(k,m))$
$O(k,m,t) = F_{a(k,m)t} + F_{b(k,m)t}$	if $t \neq node(k, m)$ and $t = node(a(k,m))$ and $t = node(b(k,m))$
$O(k,m,t) = D_{kmt}$	if $t = node(k,m) = node(a(k,m)) = node(b(k,m))$
$O(k,m,t) = D_{kmt} + E_{a(k,m)t}$	if $t = node(k,m) = node(b(k,m))$ and $t \neq node(a(k,m))$
$O(k,m,t) = D_{kmt} + E_{b(k,m)t}$	if $t = node(k, m) = node(a(k,m))$ and $t \neq node(b(k,m))$
$O(k,m,t) = D_{kmt} + E_{a(k,m)t} + E_{b(k,m)t}$	if $t = node(k,m)$ and $t \neq node(a(k,m))$ and $t \neq node(b(k,m))$
$O(k,m,t) = 0$	otherwise

where $F_{a(k,m)t}$ is the number of additional disk accesses needed at node t in order to send a(k,m) from node t to another node after having retrieved it and $E_{a(k,m)t}$ is the number of disk access required to receive and store a(k,m) at node t (typically a file write and the creation of needed indexes).

For final steps,

$O(k,m,t) = E_{a(k,m)t}$	if $t \neq$ node(a(k,m)) and t = node(k)
$O(k,m,t) = F_{a(k,m)t}$	if t = node(a(k,m)) and $t \neq$ node(k)
$O(k,m,t) = 0$	otherwise.

For update requests,

$O(k,m,t) = D_{kmt}$	if copy(a(k,m), t) = 1
$O(k,m,t) = 0$	otherwise

$$CPU(k, m) = \sum_{t} U(k,m,t)\, p_t$$

where p_t is the CPU processing cost per unit.

For message steps,

$U(k,m,t) = S_t$	if t = node(k) and $t \neq$ node(a(k,m))
$U(k,m,t) = R_t$	if $t \neq$ node(k) and t = node(a(k,m))
$U(k,m,t) = 0$	

where S_t and R_t are the expected CPU units required to send and receive a message.

For selection and projection steps,

$U(k,m,t) = W_{kmt}$	if t = node(a(k,m))
$U(k,m,t) = 0$	otherwise

where W_{kmt} is the number of CPU units required to process step m of query k at node t

For join steps,

$U(k,m,t) = F'_{a(k,m)t}$	if $t \neq$ node(k, m) and t = node(a(k,m)) and $t \neq$ node(b(k,m))
$U(k,m,t) = F'_{b(k,m)t}$	if $t \neq$ node(k, m) and $t \neq$ node(a(k,m)) and t = node(b(k,m))
$U(k,m,t) = F'_{a(k,m)t} + F'_{b(k,m)t}$	if $t \neq$ node(k, m) and t = node(a(k,m)) and t = node(b(k,m))
$U(k,m,t) = W_{kmt}$	if t = node(k, m) = node(a(k,m)) = node(b(k,m))
$U(k,m,t) = W_{kmt} + E'_{a(k,m)t}$	if t = node(k, m) = node(b(k,m)) and $t \neq$ node(a(k,m))
$U(k,m,t) = W_{kmt} + E'_{b(k,m)t}$	if t = node(k, m) = node(a(k,m)) and $t \neq$ node(b(k,m))
$U(k,m,t) = W_{kmt} + E'_{a(k,m)t} + E'_{b(k,m)t}$	if t = node(k, m) and $t \neq$ node(a(k,m)) and $t \neq$ node(b(k,m))
$U(k,m,t) = 0$	otherwise

where $F'_{a(k,m)t}$ and $E'_{a(k,m)t}$ are the number of CPU operations required to send and receive a(k,m) from and to node t, respectively.

For final steps,

$U(k,m,t) = E'_{a(k,m)t}$	if $t \neq$ node(a(k,m)) and t = node(k)
$U(k,m,t) = F'_{a(k,m)t}$	if t = node(a(k,m)) and $t \neq$ node(k)
$U(k,m,t) = 0$	otherwise.

For send-message steps of updates,

$$U(k,m,t) = \sum_{p \neq t} \text{copy}(a(k,m), p)\, S_t \text{ if } t = \text{node}(k)$$

$U(k,m,t) = R_t$	if $t \neq$ node(k) and copy(a(k,m), t) = 1
$U(k,m,t) = 0$	otherwise

For receive-message steps of updates,

$$U(k,m,t) = \sum_{p \neq t} copy(a(k,m), p)\, R_t \text{ if } t = node(k)$$

$$U(k,m,t) = S_t \qquad \text{if } t \neq node(k) \text{ and } copy(a(k,m), t) = 1$$

$$U(k,m,t) = 0 \qquad \text{otherwise}$$

For update steps,

$$U(k,m,t) = W_{kmt} \qquad \text{if } copy(a(k,m), t) = 1$$

$$U(k,m,t) = 0 \qquad \text{otherwise}$$

$$STO(t) = s_t \sum_i copy(i, t)\, L_i$$

where s_t be the unit storage cost per unit time at node t.

Appendix 2. Average Response Time

$$R_{COM}(k) = \sum_t \sum_p \sum_m \left(\frac{W(t,p)TL(t,p)N(k,m,t,p)}{(UL(t,p))^2 - UL(t,p)TL(t,p)} + \frac{H(k,m,t,p)}{UL(t,p)} \right)$$

where $UL(t,p)$ is the capacity of the communication link from node t to node p (bytes per unit time), $TL(t,p) = \sum_k f(k) \sum_m H(k,m,t,p)$, $W(t,p) = \dfrac{TL(t,p)}{\sum_k f(k) \sum_m N(k,m,t,p)}$, and $N(k,m,t,p)$ is 1 if $H(k,m,t,p) > 0$ and it is 0 otherwise.

$$R_{IO}(k) = \sum_t \sum_m O(k,m,t) \frac{1}{UIO(t) - TIO(t)}$$

where $UIO(t)$ is the disk I/O capacity at node t (number of disk I/O's per unit time) and $TIO(t) = \sum_k f(k) \sum_m O(k,m,t)$ is the total number of disk I/O's at node t.

$$R_{CPU}(k) = \sum_t \sum_m U(k,m,t) \frac{1}{UCPU(t) - TCPU(t)}$$

where $UCPU(t)$ is the CPU capacity at node t (number of instructions per unit time) and $TCPU(t) = \sum_k f(k) \sum_m O(k,m,t)$ is the total number instructions at node t.

Appendix 3. A Nested Genetic Algorithm for Distributed Database Design

Outer Genetic Algorithm:

1. Generate initial pool of solutions:
   1.a. Randomly generate a feasible data allocation (to be feasible, each file (fragment) must be allocated to at least one node),
   1.b. Use the (inner) operation allocation genetic algorithm (see below) to allocate operations for this data allocation, thus producing a complete solution for this data allocation,
   1.c. Evaluate the cost of this solution,
   1.d. Repeat until the initial solution pool is generated.

2. Iterate through successive generations:
   2.a. Probabilistically select two parent solutions from the solution pool,
   2.b. Produce a new data allocation (child) by applying crossover or mutation,
   2.c. Use the (inner) operation allocation genetic algorithm (see below) to allocate operations for this data allocation (child), thus producing a complete solution for this data allocation,
   2.d. Evaluate the cost of this solution,
   2.e. If the new solution is better than the worst solution in the solution pool, add it to the pool and remove the worst solution,
   2.f. Repeat for N generations, where N is the maximum number of iterations.

Inner Genetic Algorithm:

3. Generate initial pool of operation allocations:
   3.a. Randomly generate a feasible operation allocation for the given data allocation (to be feasible all retrieval operations must be assigned to nodes at which the required data is stored),
   3.b. Evaluate the cost of this solution,
   3.c. Repeat until the initial operation allocation pool is generated.

4. Iterate through successive generations:
   4.a. Probabilistically select two parent solutions from the operation allocation pool,
   4.b. Produce a new operation allocation (child) by applying crossover or mutation,
   4.c. Evaluate the cost of this solution,
   4.d. If the new solution is better than the worst in the operation allocation pool, add it and remove the worst,
   4.e. Repeat for M generations, where M is a maximum number of iterations.