Seoul Journal of Business
Volume 8, Number 1 (June 2002)

A Comparison of Distributed Database Design Models

Sangkyu Rho*
College of Business Administration
Seoul National University

Salvatore T. March
Owen Graduate School of Management
Vanderbilt University

Abstract

Although numerous database design models and solution algorithms have been developed, there has been little work that compares and evaluates these models. Lack of such work has left us with several questions: Do the more comprehensive models actually result in better solutions than the simpler models? If so, what makes them better? Are they better under all conditions or only under certain conditions? Are there trade-offs between data redundancy and sophisticated operation allocation strategies? In this paper, we systematically compare and evaluate several distributed database design models in terms of total operating cost and average response time under various conditions. We vary the relative frequency of update queries and selectivities of queries. The results demonstrate that replication, join node selection, join order, and reduction by semijoin, all have significant impact on the efficiency of a distributed database system. Replication was most effective for retrieval intensive and high selectivity situations. Join node selection, join order, and reduction by semijoin were most effective for balanced retrieval/update and low selectivity situations. The results also suggest that there are trade-offs between total operating cost and average response time design criteria.

* This research was partially supported by Institute of Management Research, College of Business Administration, Seoul National University (Corresponding author)

1. INTRODUCTION

With the emergence of commercial distributed database management systems [Ricciuti, 1993; Richter, 1994; The, 1994], geographically distributed database systems are becoming more common. Distributed database systems can provide users with transparent access to corporate databases that are maintained at different locations while retaining autonomy in local processing.

Properly designed distributed databases can yield significant cost and performance advantages over centralized systems for geographically distributed organizations. However, the design of a distributed database is an extremely complex process. Given a computer network consisting of nodes with given processing and storage capacities and costs, connected by links with given data transmission capacities and costs, a distributed database design must allocate data to nodes, possibly with redundancy, so that retrieval and update operations can be efficiently carried out at run time. Inappropriate placement of data or poor choices of data access or processing strategies can result in high cost and poor system performance [Edelstein, 1995a, 1995b, Ozsu and Valduriez, 1991].

The determination of units of data to allocate, termed file *fragments*, and the placement of copies of those units on nodes in the network is termed *data allocation* [Apers, 1988; Dowdy and Foster, 1982]. The choice of when, where, and how retrieval and processing operations are performed is termed *operation allocation or distributed query optimization* [Apers, Hevner, and Yao, 1983; Yu and Chaing, 1984]. A *concurrency control mechanism* [Bernstein, Shipman and Rothnie, 1980; Bernstein and Goodman, 1981] insures that update operations are performed correctly and consistently. Having an efficient and effective concurrency control mechanism is particularly important when data is allocated redundantly. Collectively, operation allocation and concurrency control are termed *operating strategies*.

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). Therefore, although operations are not actually allocated until run-time, an *ideal* operation allocation strategy must be developed at design time to obtain an effective distributed database design. Furthermore, designing an ideal operation allocation strategy enables queries to be pre-compiled and allows designers to optimize the overall system performance instead of optimizing the performance of singe queries in isolation [Drenick and Smith, 1993]. The concurrency control mechanism is typically dependent upon the selected distributed database management system.

Numerous distributed database design models and solution algorithms have been developed. Apers [1988], Blankinship et al. [1991], and Cornell and Yu [1989], developed models for the combined data and operation allocation problem. However, none of this research considers the effects of data replication or update queries and the requisite concurrency control mechanisms.

Ram and Narasimhan [1994] and March and Rho [1995] developed models which include data replication and a concurrency control mechanism. However, they ignore important aspects of operation allocation such as data reduction by semijoins, a major focus of distributed query optimization research [Apers, et al., 1983, Hevner and Yao, 1979, Yoo and Lafortune, 1989], and the determination of join order, a known determinant of query processing efficiency [Mishra and Eich, 1992]. Rho and March [2000] extend the approach of March and Rho [1995] to include these.

Although numerous models have been developed, there has been little work that compares and evaluates these models. Lack of such work has left us with several questions: Do the more comprehensive models actually result in better solutions than the simpler models? If so, what makes them better? Are they better under all conditions or only under certain conditions? Are there trade-offs between data redundancy and sophisticated operation allocation strategies? Systematic comparison of different models under various conditions can provide us with insights into such questions.

In this paper, we systematically compare and evaluate several distributed database design models. We compare designs produced by models based on the work of Ram and Narasimhan [1994], Cornell and Yu [1989], March and Rho [1995], and Rho and March [2000], in terms of total operating cost and average response time under various conditions. We vary the relative frequency of update queries and selectivities of queries. The results demonstrate that replication, join node selection, join order, and reduction by semijoin, all have significant impact on the efficiency of a distributed database system. Replication was most effective for retrieval intensive and high selectivity situations. Join node selection, join order, and reduction by semijoin were most effective for balanced retrieval/update and low selectivity situations. Their combination was effective under all conditions, although the additional contribution of each one given the other was often limited. That is, when replication was effective, the additional contribution of the sophisticated operation allocation strategies was very limited. Conversely, when sophisticated operation allocation strategies were effective, the additional contribution of replication was very limited.

The remainder of the paper is organized as follows. In the next section, we describe and classify the above mentioned distributed database design models. In the following section, we briefly present an example problem and its variations solved using each model. We then present a comparison of the results. Finally, we discuss the insights gained in this research.

2. DISTRIBUTED DATABASE DESIGN MODELS

In this section, we first describe two aspects of distributed database design: data allocation and operation allocation. We then describe how different models treat each aspect. This produces a taxonomy of models for comparison and evaluation.

2.1 Data Allocation

Data allocation includes the determination of units of data to allocate, termed *fragments*, and the placement of copies of those units on nodes in the network. Data allocation produces a

subschema for each node of the distributed database system [Ceri et al., 1987]. The process of determining units of data to allocate is termed *fragmentation* [Navathe et al., 1984]. There are two types of fragmentations, horizontal and vertical. Horizontal fragmentation groups records that satisfy a selection condition [Ozsu and Valduriez, 1991b]. Vertical fragmentation groups attributes 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 reference it allows all retrieval queries to be processed locally. However, such data replication increases the operating cost and response time of update queries since all copies of the affected 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, 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. Each is described briefly below. *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 required by the query is selected for processing.

Reduction applies only to join queries when the fragments to be joined are stored at different nodes. In it, *semijoins* [Bernstein and Chiu, 1981] are used to reduce the amount of data that must be transferred to accomplish join operations. To join two fragments stored at different nodes, the required data from one of the fragments must be transmitted to the node at which the other is stored, or the required data from both must be transmitted to a third join node. However, if there are records in one fragment without corresponding records in the other fragment data can be transmitted unnecessarily. The use of

semijoins eliminates this unnecessary data transmission by identifying the records that have matching join values before they are transmitted as follows.

When two fragments are joined using a semijoin, one is designated as the *reducer* and the other as the *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 if its join attribute matches one of the transmitted join values. Only the selected records of the reducee are transmitted to the node at which the join is performed, typically the reducer node. A semijoin is effective if the cost of transmitting the join attribute values from the reducer node to the reducee node is exceeded by the reduction in the cost obtained by transmitting only selected records rather than all records from the reducee node to the join node. Determining if semijoins are effective is a complex task [Yoo and Lafortune, 1989].

In *assembly*, data are sent to the result node, if they are not already there, and final processing such as sorting and aggregations is performed. In much of the research on distributed query optimization it is assumed that all reduced fragments are sent to the result node where all joins are performed in some predetermined order. Hence, the assembly phase included all join operations. In this research the solution algorithms determine the nodes at which joins are performed and the order in which joins are performed.

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 et al., 1986], timestamp-based [Bernstein et al., 1980], optimistic [Ceri and Owicki, 1981]). Two-phase locking (2PL) is the most commonly implemented. Distributed 2PL, one variation of 2PL, is used in this research [Ram and Narasimhan, 1994].

2.3 Types of Models

In order to systematically compare distributed database design models, we classify them based on their data and operation

allocation strategies. We consider two types of data allocation strategies: no replication (NR) and replication (R). In the no replication strategy, each fragment is allocated to exactly one node, while in the replication strategy, each fragment may be allocated to more than one node. We consider three types of operation allocation strategies. Operation allocation strategy 1 (OA1) is based on the work of Ram and Narasimhan [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 semijoin and assumes that joins are performed in a predetermined order. It is based on the work of Cornell and Yu [1989] and March and Rho [1995]. Finally, operation allocation strategy 3 (OA3) integrates copy identification, *beneficial* semijoin[1] selection, join node selection, and join ordering. It is the model presented in Rho and March [2000]. The combination of two data allocation strategies and three operation allocation strategies results in six different distributed database design models (see Figure 1.). The distributed database design models of Ram and Narasimhan [1994], Cornell and Yu [1989], March and Rho [1995], and Rho and March [2000] correspond to R-OA1, NR-OA2, R-OA2, and R-OA3, respectively. NR-OA1 is used as a base case and NR-OA3 is used to study the effects of replication for the comprehensive operation allocation strategy, OA3.

We solved a set of problems for each model described above. Each problem was solved twice, once using a total operating cost minimization objective[2] and once using an average response time minimization objective. Formulations for each objective are summarized in Appendices 1 and 2, respectively. They are

1) A semijoin is beneficial when its benefit (e.g., reduction in the cost of transmitting the reducee) outweighs its cost (e.g., cost of transmitting the join attributes of the reducer).

2) Minimizing cost essentially results in a weighted minimization of required system resources, an important consideration in establishing and conforming a distributed information system budgetary requirements. Presumably, minimizing the computing resources required by this distributed database makes those resources available for other applications and/or makes the purchase of additional resources unnecessary [Rho and March, 2000].

Operation Allocation Strategy	Data Allocation Strategy	
	No Replication (NR)	Replication (R)
Copy Identification (OA1)	NR-OA1 (Base Case)	R-OA1 (Ram & Narasimhan, 1994)
+Join Node Selection (OA2)	NR-OA2 (Cornell & Yu, 1989)	R-OA2 (March and Rho, 1995)
+Beneficial Semijoin & Join Order (OA3)	NR-OA3	R-OA3 (Rho and March, 2000)

Figure 1. Types of Distributed Database Design Models

described in detail in Rho [1995]. The next section describes the example problem and its variations used in this study. The following section discusses our experimental results.

3. EXPERIMENTAL PROBLEMS

3.1 Basic Problem Description

Consider an order processing system for a distribution company.[3] Suppose the company has 4 regional offices (Regions 1 through 4) and a headquarters (HQ) each having the same disk and CPU capacities and unit costs (e.g., 400 IOs/sec, $2.50/M IOs; 20 MIPS, $0.00005/MIPS; 1 Gbytes/disk, $10.00/Mbytes/month). Further, suppose that these are in a fully connected network, each link having the same capacity and unit cost (e.g., 56 Kbits/sec, $2.00/Mbytes).

Each regional office serves customers in that region. A customer places an order through a particular salesperson for a given quantity of a specific product that is to be shipped by a certain shipping date. Once the order is filled it is saved for future reference. The system keeps track of orders, customers, and salespeople and provides information for management decision making.

The database for this application has four relations: Salesperson, Customer, Order, and Product (Figure 2). Ten types

3) The problem was created based on a real database system. It was simplified to ensure understandability and computablility.

Salesperson (1,200 instances)

sales_id	Text	6
sales_name	Text	20
addr	Text	30
phone	Text	10
reg_no	Text	2

Product (10,000 instances)

prod_no	Text	10
prod_type	Text	15
prod_name	Text	20
price	Numeric	10.2
quantity_on_hand	Numeric	10.2

Customer (240,000 instances)

cust_id	Text	8
cust_name	Text	20
addr	Text	30
phone	Text	10
company	Text	20
reg_no	Text	2
sales_id	Text	6

Order (2,020,000 instances)

order_no	Text	12
date	Date	6
cust_id	Text	8
sales_id	Text	6
prod_no	Text	10
quantity	Text	5
total_price	Text	10.2
shipping_date	Date	6
date_filled	Date	6

Figure 2. An Example Database

of retrieval queries and five types of update queries are executed with varying frequencies and selection criteria at different nodes (see Appendix 3 and Table 1).

Table 2 shows the fragments derived from the selection criteria stated in these queries plus implicit selection criteria specified for their execution (See, e.g., Ozsu and Valduriez [1991b]). Implicit selection criteria limit the values for specified parameters in the queries. The values for order_no specified in R1, for example, are limited to unfilled orders for customers in the region from which the query originates. The Salesperson and Customer relations are fragmented by regions. The Product relation is not fragmented since each query needs to access the whole relation. The Order relation is first fragmented into unfilled and filled order fragments. Each of these is further fragmented by region.

Considering differences in explicit and implicit selection criteria, regional offices and headquarters execute a total of 37

Table 1. Frequencies of Retrieval and Updates from Each Node

Query	Origination Node				
	HQ	Region 1	Region 2	Region 3	Region 4
R1		1000	400	600	800
R2	400				
R3	400	200	80	80	80
R4		200	80	120	200
R5		200	80	120	200
R6		200	200	120	200
R7	200				
R8		200	80	120	160
R9		4000	1600	2400	3200
R10	400	4000	1600	2400	3200
U1	3000				
U2		2000	500	1500	2000
U3		40000	10000	20000	40000
U4		20000	5000	15000	20000
U5		40000	10000	30000	40000

Table 2. File Fragments (Instances)

Relation	Fragments			
Salesperson (1.2K)	S1 (.45K)	S2 (.15K)	S3 (.25K)	S4 (.35K)
Customer (240K)	C1 (90K)	C2 (30K)	C3 (50K)	C4 (70K)
Order (2,020K)	UO1 (7K)	UO2 (3K)	UO3 (4K)	UO4 (6K)
	FO1 (700K)	FO2 (300K)	FO3 (400K)	FO4 (600K)
Product (10K)	P (10K)			

retrieval queries and 17 update queries. Table 3 and Table 4 show the frequencies with which the fragments are accessed at each node for retrieval and update queries, respectively. These frequencies are derived from Table 1 based on origination nodes and selection criteria of the queries.

3.2 Problem Variations

To assess the performance of each model in a variety of environments, we varied the characteristic of the above problem along two dimensions: query mix and query selectivity.

Table 3. Frequency of Fragment Access (R1-R10)

Query*		Fragments	HQ	Region 1	Region 2	Region 3	Region 4
R1	R1.1	C1, UO1, P		1000			
	R1.2	C2, UO2, P			400		
	R1.3	C3, UO3, P				600	
	R1.4	C4, UO4, P					800
R2	R2.1	S1, C1, FO1	100				
	R2.2	S2, C2, FO2	100				
	R2.3	S3, C3, FO3	100				
	R2.4	S4, C4, FO4	100				
R3	R3.1	FO1, P	400	200			
	R3.2	FO2, P	400		80		
	R3.3	FO3, P	400			80	
	R3.4	FO4, P	400				80
R4	R4.1	C1, UO1		200			
	R4.2	C2, UO2			80		
	R4.3	C3, UO3				120	
	R4.4	C4, UO4					200
R5	R5.1	S1, UO1		200			
	R5.2	S2, UO2			80		
	R5.3	S3, UO3				120	
	R5.4	S4, UO4					200
R6	R6.1	S1, C1		200			
	R6.2	S2, C2			200		
	R6.3	S3, C3				120	
	R6.4	S4, C4					200
R7	R7.1	UO1	200				
	R7.2	UO2	200				
	R7.3	UO3	200				
	R7.4	UO4	200				
R8	R8.1	S1		200			
	R8.2	S2			80		
	R8.3	S3				120	
	R8.4	S4					160
R9	R9.1	C1		4000			
	R9.2	C2			1600		
	R9.3	C3				2400	
	R9.4	C4					3200
R10	R10	P	400	4000	1600	2400	3200

Table 4. Frequency of Fragment Access (U1-U5)

Query		Fragments	HQ	Region 1	Region 2	Region 3	Region 4
U1	U1.1	S1	1000				
	U1.2	S2	250				
	U1.3	S3	750				
	U1.4	S4	1000				
U2	U2.1	C1		2000			
	U2.2	C2			500		
	U2.3	C3				1500	
	U2.4	C4					2000
U3	U3.1	UO1		40000			
	U3.2	UO2			10000		
	U3.3	UO3				20000	
	U3.4	UO4					40000
U4	U4.1	FO1		20000			
	U4.2	FO2			5000		
	U4.3	FO3				15000	
	U4.4	FO4					20000
U5	U5	P		40000	10000	30000	40000

Query Mix. Query mix is the relative execution frequency of retrieval compared to update queries. The primary design decision affected by query mix is replication. In an update intensive application, the retrieval advantages of replication can be outweighed by increases in update costs as all the copies of the data must be updated (how they are updated depends on the concurrency control mechanism). Three classes of query mix are considered: Balanced (B), Retrieval Intensive (RI), and Update Intensive (UI). Frequencies for the balanced problem are shown in Table 3 and Table 4.

Query Selectivity: Query selectivity is the proportion of records selected by a query. When large proportions of the records are selected by retrieval queries, semijoins may not be beneficial. Two environments are considered: Low Selectivity (L) and High Selectivity (H). In a low selectivity environment, most retrieval queries require small proportions of the records (selectivities between .05 and .25). In a high selectivity environment, most retrieval queries require large proportions of the records (selectivities between .75 and .90).

This results in 6 variations of the example problem as shown

Table 5. Problem Variations

Problem	Query Mix	Query Selectivity
P1	B	L
P2	B	H
P3	RI	L
P4	RI	H
P5	UI	L
P6	UI	H

in Table 5. We solved each variation once to minimize total operating cost and once to minimize average response time for each model. The example problem has approximately 4.1×10^{156} possible solutions. Although some of these are infeasible due to constraints, exhaustive enumeration is not a viable solution method. Furthermore, due to the nature of the dependencies among decision variables, standard mathematical models such as integer programming are also infeasible. Hence, we have adopted the nested genetic algorithm developed in Rho [1995]. While the genetic algorithm cannot guarantee optimality, it consistently generates extremely good solutions in a reasonable amount of computer time [March and Rho, 1995].

The genetic algorithm is written in C++ and runs in UNIX or PC environments. Its run time depends on problem size and on algorithm parameters. The same problem and genetic algorithm parameters were used for the entire set of experiments to ensure comparability. We discuss the results of these experiments in the next section.

4. EXPERIMENTAL RESULTS

We first solved each problem variation to minimize total operating cost. Average response time was calculated for each solution. The minimum total operating costs and average response times for each model for each problem variation are shown in the first set of columns of Appendix 4. We then solved each problem to minimize average response time. Total operating cost was calculated for each solution. These are also shown in the second set of columns of Appendix 4.

4.1 Minimum Total Operating Cost

Figure 3 illustrates the overall effects of the different distributed database design models when the design objective is to minimize total operating costs. Operating costs and response times are averaged over all problem variations and reported relative to the base case NR-OA1. In Figure 3, each bar represents the average total operating cost of the best solution found for the model specified. Dotted lines represent the average response time for those solutions.

As shown in Figure 3, replication reduced the minimum operating cost significantly across all of the operation allocation strategies. However, its effect became smaller as the operation allocation strategy became more comprehensive (a 30% reduction for OA1, an 18% reduction for OA2, and a 12 % reduction for OA3). Join node selection (OA2) also reduced the cost significantly. Its effect was more significant when replication was not allowed (a 24% reduction without replication and a 11% reduction with replication). Semijoins and join order (OA3) further reduced the minimum operating cost significantly.

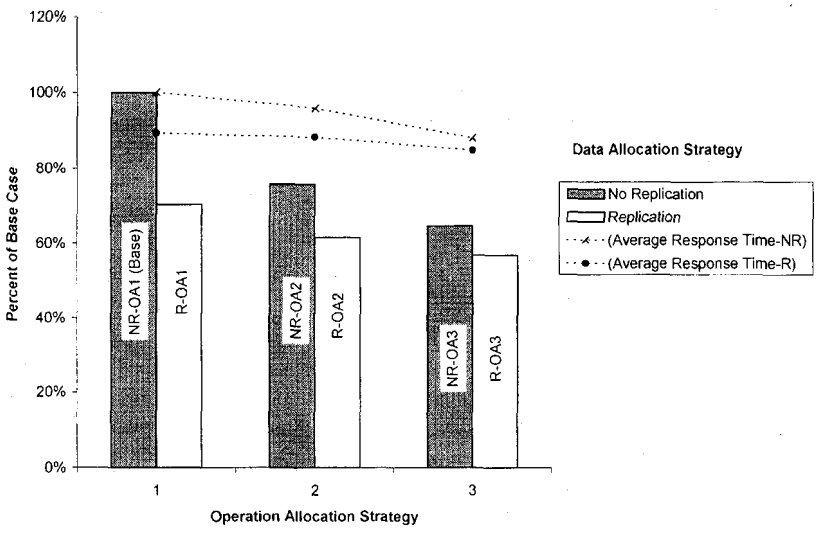

Figure 3. Effects of Data and Operation Allocation Strategies on the Minimum Total Operating Cost [Overall]

Like join node selection, their effect was more significant when replication was not allowed (a further reduction of 11% without replication and 5% with replication). As expected, the average response time was reduced as the minimum total operating cost was reduced (the dotted lines in Figure 3.).

The comprehensive model proposed by Rho and March [2000] outperformed all other models including those by Ram and Narasimhan [1994] (R-OA1) and Cornell and Yu [1989] (NR-OA1). Interestingly, NR-OA3 slightly outperformed Ram & Narasimhan [1994]. This suggests that sophisticated operation allocation strategies can be as effective as data replication in reducing system operating costs under certain conditions. Furthermore, given a design with replication or sophisticated operation allocation strategies, adding the other does not significantly contribute to performance. In the following, we discuss the results of two contrasting cases: retrieval intensive problems with high selectivity and update intensive problems with low selectivity, which are characteristic of decision support systems and on-line transaction systems, respectively.

In retrieval intensive problems with high selectivity, the effect of replication was much more significant than that of comprehensive operation allocation strategies. As shown in Figure 4, replication significantly reduced the minimum total

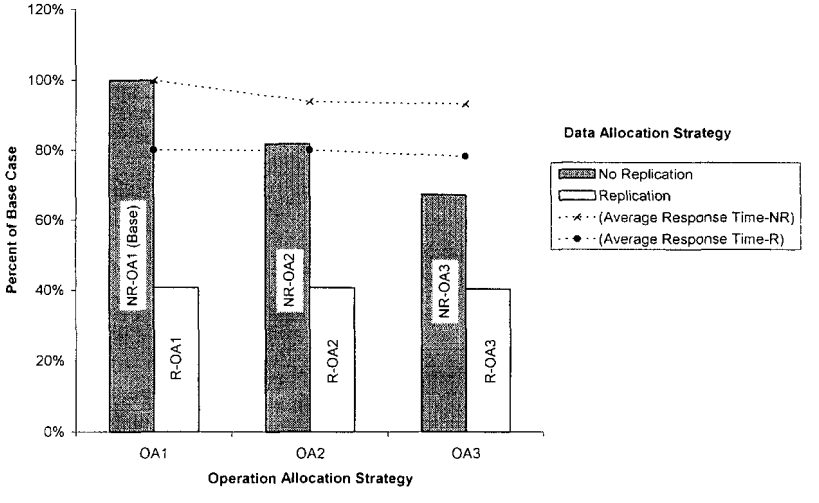

Figure 4. Effects on the Minimum Total Operating Cost of P4 (RI, H)

operating cost across different operation allocation strategies. Join node selection (OA2) and reduction by semijoin and join order (OA3) reduced the minimum total operating cost significantly only when replication was not allowed, and even then not as effectively as replication alone. This is expected since sophisticated operation allocation strategies generally reduce the amount of data transmission needed to process a query while replication eliminates the need to transmit data at all. If little or no data is transmitted, then sophisticated operation allocation strategies will not be very effective. In the extreme case when there is no update and data storage is relatively inexpensive, self contained nodes (i.e., nodes that contain a copy of all data needed at that node) are, in fact, optimal and operation allocation is irrelevant.

In update intensive problems with low selectivity, the opposite is true. As shown in Figure 5, the effects of join node selection, reduction by semijoin, and join order are much more significant than those of replication. Replication moderately reduced the minimum cost when only copy identification (OA1) was used but did not significantly reduce the cost for the other operation allocation strategies. This is reasonable since the problem is update intensive. Replication will lead to more increases in update costs in an update intensive problem than in a retrieval

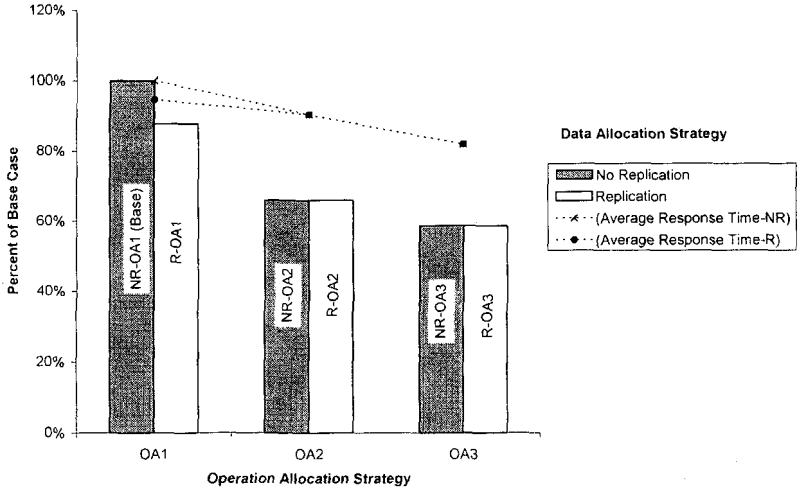

Figure 5. Effects on the Minimum Total Operating Cost of P5 (UI, L)

intensive problem, which makes replication less effective. Therefore, it is likely that fewer copies will be replicated in an update intensive problem.

4.2 Minimum Average Response Time

There are subtle interactions between response time and operating costs. Generally the minimum cost solution does not also minimize the average response time and conversely. We solved a set of the problems discussed above to minimize average response time rather than total operating cost. The minimum average response times and total operating costs of the solutions for each model are reported in Appendix 4 and summarized in Figure 6. Opposite of Figure 3, bars in Figure 6 represent the average response time of the best solutions found for each model while the dotted lines represent the total operating cost for those solutions. Similar to Figure 3 these are reported relative to the base case NR-OA1.

As shown in Figure 6, data replication and operation allocation strategies have similar effects on the minimum average response time. However, their effects are not as significant as on the minimum operating cost.

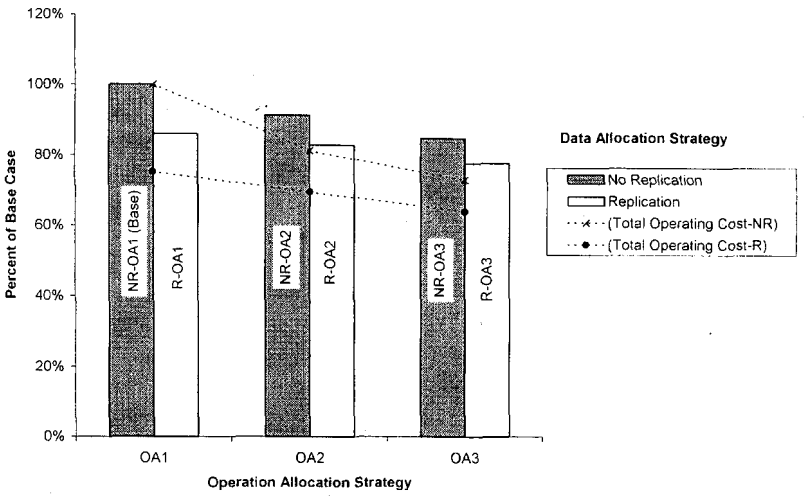

Figure 6. Effects of Data and Operation Allocation Strategies on the Minimum Average Response Time [Overall]

Replication moderately reduced the minimum average response time across all of the operation allocation strategies. Its effect became smaller as the operation allocation strategy became more comprehensive (a 14% reduction for OA1, a 9% reduction for OA2, and an 8 percent reduction for OA3). Join node selection (OA2) also moderately reduced the minimum average response time. As expected, its effect was more significant when replication was not allowed (a 9% reduction without replication and 4% reduction with replication). Semijoins and join order (OA3) further reduced the minimum average response time (a further reduction of 7% without replication and 6% with replication).

The total operating costs were reduced as the minimum average response time was reduced. Models with more sophisticated operation allocation strategies outperformed prior models in terms of average response time as well as total operating costs. As in the minimum total operating cost cases, NR-OA3 slightly outperformed R-OA1 [Ram and Narasimhan, 1994]. All but the base-case outperformed NR-OA2 [Cornell and Yu, 1989]. This suggests that operation allocation strategies can be as effective as data replication in reducing system response time as well as in reducing system operating costs. In the following we discuss the results for two contrasting problem in more detail.

In retrieval intensive problems with high selectivity, the effects of replication were more significant than those of comprehensive operation allocation strategies. As shown in Figure 7, replication significantly reduced the minimum average response time across different operation allocation strategies. Join node selection (OA2) and reduction by semijoin and join order (OA3) reduced the minimum response time significantly only when replication was not allowed. They slightly reduced the response time when replication was allowed. This problem illustrates that reductions in the average response time do not necessarily result in reductions in the total operating cost. In this case join node selection (OA2) afforded reductions in the average response time but increased the total operating cost. This is explained by the fact that less expensive nodes or links may become congested, resulting in poor overall system response time. Join node selection enabled more balanced loads across the nodes but

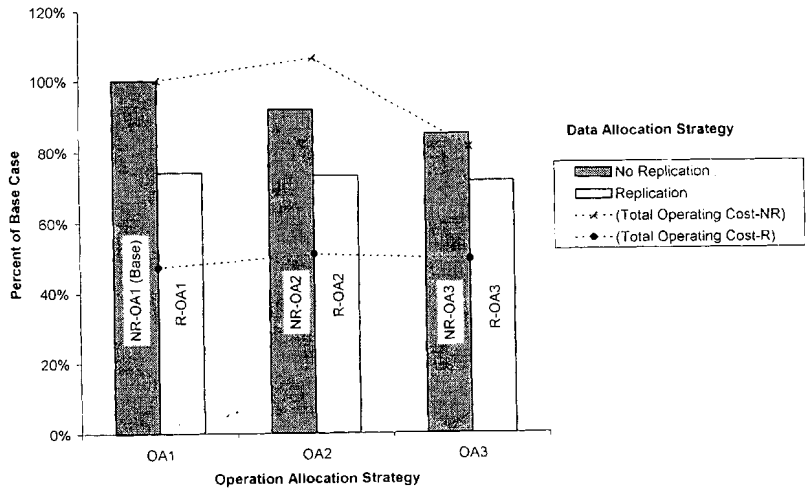

Figure 7. Effects on the Minimum Average Response Time of P4 (RI, H)

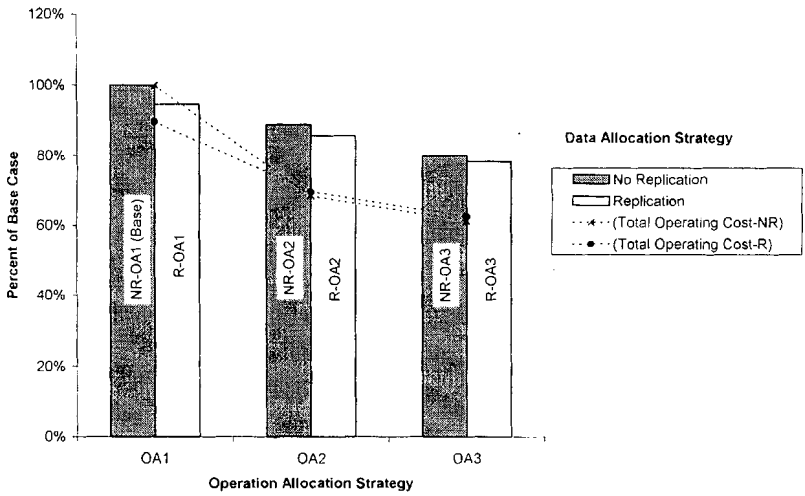

Figure 8. Effects on the Minimum Average Response Time of P5 (UI, L)

resulted in higher operating costs.

In update intensive problems with low selectivity, the effects of comprehensive operation allocation strategies were much more significant than those of replication. As shown in Figure 8, replication moderately reduced the minimum response time

when only copy identification (OA1) was used and slightly reduced the response time for the other operation allocation strategies. In contrast, join node selection and semijoins and join order significantly reduced the minimum average response time both with and without replication. This supports our contention that for low selectivity problems, efficient operation allocation strategies are an effective way to gain efficiency. For OA2 and OA3, slight reductions in the average response times resulted in slight increases in the total operating costs.

4.3 Discussion

In our experiments, replication was effective for retrieval intensive problems with either high or low retrieval selectivities. It was generally not effective for update intensive problems. This is reasonable since replication gains its retrieval efficiencies by storing multiple copies of data where they are used. It thus reduces communication costs, but increases update and storage costs. Its efficiency is not dependent upon the selectivities of the retrievals.

Comprehensive operation allocation strategies were effective for low retrieval selectivity problems, independent of the retrieval/update mix. They were only moderately effective for high selectivity problems (less so than replication). This is reasonable since operation allocation strategies focus on processing the existing data in the most efficient manner. Since they do not depend on data replication to be effective, update costs are generally not effected. However, if join queries have high selectivities, semijoins are not as effective and join order will have limited effect.

Considering problem variations, the following conclusions are drawn for the problems studied. For retrieval intensive problems with high retrieval selectivities, replication is more effective than operation allocation strategies. Their combination offers little, if any, improvement over replication alone. For update intensive problems with low retrieval selectivities, operation allocation strategies are more effective than replication. Their combination offers little improvement.

To be effective over the widest range of problems, a distributed database design model must include both replication and a

comprehensive set of operation allocation strategies. Hence, replication can be selected when it is efficient, and appropriate processing strategies can be used to determine a globally efficient design.

Average response times were often reduced when the minimum total operating costs were reduced. This is reasonable since less processing and communication (i.e., lower cost) is likely to result in lower response time. However, the total operating costs were not necessarily reduced when the minimum average response times were reduced. This is explained by the fact that processing loads should be balanced in order to reduce the average response time. Congested nodes or links may be cost effective but are likely to result in poor overall system response time. Therefore, rather than congesting less expensive nodes or links, the minimization of average response time solution would balance loads which could result in higher operating costs.

5. SUMMARY AND FUTURE RESEARCH

In this paper, we systematically compared distributed database design models, including models posed by Ram and Narasimhan [1994], Cornell and Yu [1989], March and Rho [1995], and Rho and March [2000] in terms of total operating cost and average response time under various conditions. The results demonstrate that replication, join node selection, join order, and reduction by semijoin have significant impact on the performance of a distributed database system. Replication was effective for retrieval intensive and high selectivity problems. Join node selection, join order, and reduction by semijoin were effective for balanced retrieval/update and low selectivity problems. Their combination typically offered only marginal improvement or, at best, moderate improvement, over the more effective strategy. The results also suggest that there are trade-offs between total operating cost and average response time.

Comparison of models provides us with insights into the effects of different data and operation allocation strategies under various conditions. Such insights can be valuable for designers of distributed databases and for organizations who must purchase or develop a distributed database management

system. If, for example, an application is known to be update intensive, the designer may decide to avoid replication. This reduces the complexity of the design process, greatly simplifying the task. If a majority of applications are update intensive, the organization may decide to purchase a DBMS that supports a wide range of operation allocation strategies in its query optimizer rather than one that supports replication.

There are several major areas for future research. First, the effects of data and operation allocation strategies on the efficiency of distributed database systems should be further analyzed under various conditions using real business problems. These include different types of networks with different performance parameters such as wide area networks (WAN), local area networks (LAN), and asynchronous transfer mode (ATM) networks and different types of concurrency control mechanisms such as primary copy 2PL and asynchronous updates (e.g., store and forward).

Second, the models must be evaluated and verified in a realistic environment. Selected solutions should be implemented and their performance measured in a real organizational settings. Third, the effects of parallelism must be studied. Processing data on different nodes in parallel can significantly reduce the response time for a query. Current response time models simply add up the processing times and delays for all components of each query. Finally, a systematic sensitivity analysis should be performed to further define distributed database design guidelines and rules of thumb.

6. REFERENCES

Apers, P. M. G., "Data Allocation in Distributed Database Systems," *ACM Transactions on Database Systems*, Vol. 13, No. 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*, Vol. SE-9, No. 1, January 1983, pp. 57-68.

Bernstein, P. A. and Chiu, D. W., "Using Semi-Joins to Solve Relational Queries," *Journal of the ACM*, Vol. 28, No. 1, January 1981, pp. 25-40.

Bernstein, P. A. and Goodman, N., "Concurrency Control in Distributed

Database Systems," *ACM Computing Surveys*, Vol. 13, No. 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*, Vol. 5, No. 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 17th 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 6th Berkeley Workshop on Distributed Data Management and Communication Networks*, Berkeley, CA, February 1982, pp. 117-130.

Ceri, S., Pernici, B., and Wiederhold, G., "Distributed Database Design Methodologies," *Proceedings of the IEEE*, Vol. 75, No. 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*, Vol. 15, No. 8, August 1989, pp. 1004-1009.

Dowdy, L. W. and Foster, D. V., "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, Vol. 14, No. 2, June 1982, pp. 287-314.

Drenick, P. E. and Smith, E. J., "Stochastic Query Optimization in Distributed Databases," *ACM Transactions on Database Systems*, Vol. 18, No. 2, June 1993, pp. 262-288.

Edelstein, H., "The Challenge of Replication," *DBMS*, March 1995a, pp. 46-49, 52.

Edelstein, H., "The Challenge of Replication, Part 2," *DBMS*, April 1995b, pp. 62-70, 103.

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

Hevner, A. R. and Yao, S. B., "Query Processing in Distributed Database Systems," *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 3, May 1979, pp. 177-187.

March, S. T., "Techniques for Structuring Database Records," *ACM Computing Surveys*, Vol. 15, No. 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*, Vol. 7, No. 2, April 1995, pp. 305-317.

Mishra, P. and Eich, M. H., "Join Processing in Relational Databases," *ACM Computing Surveys*, Vol. 24, No. 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, Vol. 11, No. 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*, Vol. 9, No. 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*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1991b.

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

Ram, S. and Narasimhan, S., "Database Allocation in a Distributed Environment: Incorporating a Concurrency Control Mechanism and Queuing Costs," *Management Science*, Vol. 40, No. 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 Decision Support Tool for Distributed Database Design," *Seoul Journal of Business*, Vol. 6, No. 1/2, December 2000, pp. 71-111.

Ricciuti, M., "DBMS Vendors Chase Sybase for Client/Server," *Datamation*, Vol. 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*, Vol. 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*, Vol. 1, No. 2, June 1989, pp. 226-237.

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

Appendix 1. Total Operating Cost

$$Min \quad Cost = \sum_{k} f(k) \sum_{m} (COM(k,m) + IO(k,m) + CPU(k,m)) + \sum_{t} STO(t)$$

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 below.

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

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,

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

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

For receive-message steps of updates,

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

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

For update steps,
$U(k,m,t) = W_{kmt}$ if copy($a(k,m)$, t) = 1
$U(k,m,t) = 0$ otherwise

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

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

Appendix 2. Average Response Time

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

where $R_{COM}(k)$, $R_{IO}(k)$, and $R_{CPU}(k)$ are the times spent by query k in communication, disk I/O, and CPU, respectively. These response time components are summarized below.

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

where $UL(t,p)$ is the capacity of the communication link from node t to node p (bytes per unit time), $TL(t,p) = \sum_k f(k) \sum_m H(k,m,t,p)$, $W(t,p) = \frac{TL(t,p)}{\sum_k f(k) \sum_m N(k,m,t,p)}$, 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.

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

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. Queries

Retrieval Queries

R1. Print invoices

```
SELECT   order_no, date, prod_no, prod_name, quantity,
         price, cust_id, cust_name, customer.addr,
         customer.phone, sales_id
FROM     customer, order, product
WHERE    customer.cust_id = order.cust_id
         AND order.prod_no = product.prod_no AND
         order_no = [specified]
```

R2. Orders by customer and salesperson

```
SELECT   sales_id, sales_name, cust_id, cust_name,
         order_no, date, prod_no, quantity, price
FROM     salesperson, customer, order
WHERE    order.sales_id = salesperson.sales_id
         AND customer.cust_id = order.cust_id
         AND salesperson.reg_no = [specified]
         AND date_filled IS NOT NULL
```

R3. New Orders by product

```
SELECT   prod_no, prod_type, prod_name, count(order_no),
         sum(quantity)
FROM     order, product
WHERE    order.prod_no = product.prod_no AND date
         > [specified]
         AND date_filled IS NOT NULL
```

R4. Unfilled orders by customer

```
SELECT   cust_id, cust_name, order_no, date,
         prod_no, quantity,
         price, shipping_date
FROM     customer, order
WHERE    customer.cust_id = order.cust_id AND
         date_filled IS NULL
         AND reg_no = [specified]
```

R5. Unfilled orders by salesperson

```
SELECT    sales_id, sales_name, order_no, date,
          prod_no, quantity, price,
          shipping_date
FROM      salesperson, order
WHERE     salesperson.sales_id = order.sales_id AND
          date_filled IS NULL
          AND reg_no = [specified]
```

R6. Customers of a salesperson

```
SELECT    reg_id, sales_id, sales_name, cust_id,
          cust_name
FROM      salesperson, customer
WHERE     salesperson.sales_id = customer.sales_id
          AND sales_name = [specified]
```

R7. Unfilled orders past their shipping dates

```
SELECT    prod_no, order_no, quantity, shipping_date
FROM      order
WHERE     shipping_date > [today] AND date_filled IS
          NULL
```

R8. Salespersons

```
SELECT    sales_id, sales_name, addr, phone, reg_no
FROM      salesperson
WHERE     sales_id = [specified]
```

R9. Customers

```
SELECT    cust_id, cust_name, addr, phone
FROM      customer
WHERE     cust_id = [specified]
```

R10. Products

```
SELECT    prod_no, prod_type, prod_name, price,
          quantity_on_hand
FROM      product
WHERE     prod_no = [specified]
```

Update Queries

U1. Maintain salesperson data

```
UPDATE      salesperson
SET         reg_no = [specified], addr = [specified],
            phone = [specified]
WHERE       sales_id = [specified]
```

U2. Add a new customer

```
INSERT INTO  customer
VALUES       ('cust_id', ..... , '')
```

U3. Place a new order (including referential integrity check)

```
INSERT INTO  order
VALUES       ('order_no', ..... , '')
```

U4. Mark an order as filled

```
UPDATE      order
SET         date_filled = [specified]
WHERE       order_no = [specified]
```

U5. Adjust inventory

```
UPDATE      product
SET         quantity_on_hand = [specified]
WHERE       prod_no = [specified]
```

Appendix 4. Minimum Total Operating Cost and Minimum Average Response Time

Problem	Data Allocation Strategy	Operation Allocation Strategy	Performance Objective			
			Minimize Total Operating Cost		Minimize Average Response Time	
			Operating Cost	Response Time	Operating Cost	Response Time
P1	NR	OA1	7269.18	7.99	7269.18	7.99
	NR	OA2	4215.67	7.73	4565.76	6.60
	NR	OA3	3675.94	6.71	4076.43	6.02
	R	OA1	6028.63	7.32	6306.80	6.95
	R	OA2	4002.25	6.98	4860.79	6.48
	R	OA3	3641.67	6.68	4132.80	5.58
P2	NR	OA1	9649.91	12.12	9649.91	12.12
	NR	OA2	8734.58	11.96	8734.58	11.96
	NR	OA3	7320.66	11.64	10271.10	11.50
	R	OA1	6837.76	10.85	7844.73	10.32
	R	OA2	6731.20	11.02	8525.73	10.17
	R	OA3	6273.73	11.26	7454.15	9.81
P3	NR	OA1	7414.57	50.42	7977.42	50.30
	NR	OA2	4353.54	47.67	4720.07	42.96
	NR	OA3	3484.04	38.91	3754.42	39.08
	R	OA1	3686.59	42.82	4479.76	40.32
	R	OA2	3686.59	42.82	4043.32	39.02
	R	OA3	3390.37	39.63	3938.83	36.09
P4	NR	OA1	11154.50	79.05	11154.50	79.05
	NR	OA2	9134.05	74.30	11845.50	72.60
	NR	OA3	7498.43	73.76	9051.07	67.09
	R	OA1	4548.62	63.28	5262.41	58.52
	R	OA2	4548.62	63.28	5661.40	57.84
	R	OA3	4508.06	61.86	5501.11	56.55
P5	NR	OA1	4834.40	4.32	4834.40	4.32
	NR	OA2	3185.87	3.91	3304.42	3.84
	NR	OA3	2839.70	3.54	2958.07	3.46
	R	OA1	4245.17	4.09	4333.02	4.09
	R	OA2	3185.87	3.91	3366.02	3.70
	R	OA3	2836.06	3.54	3016.69	3.39
P6	NR	OA1	5663.73	5.62	5663.73	5.62
	NR	OA2	5663.73	5.62	5663.73	5.62
	NR	OA3	4989.90	5.37	4989.90	5.37
	R	OA1	5097.13	5.34	5097.13	5.34
	R	OA2	4979.83	5.37	5097.13	5.34
	R	OA3	4618.66	5.25	4996.32	5.21