Proceedings of the Twentieth Annual Hawaii International Conference on System Sciences, 1987.

# DESIGN AND PERFORMANCE STUDY OF DDBLMN: A DISTRIBUTED DATABASE ON A LOCAL COMPUTER SYSTEM

*Yao-Nan Lien* * *and Benjamin W. Wah* **

## ABSTRACT

DDBLMN is a distributed database on a local computer system connected by a multiaccess/broadcast network. Concurrency control, distribution design, and query processing are the three important problems studied in DDBLMN. A broadcast bus allows information to be distributed quickly and efficiently, and hence simplifies the solutions to some of these problems. A transaction processing model that integrates the control strategies in concurrency control and query processing is proposed. Based on the special characteristics of local multiaccess networks, the dynamic query processing algorithm is realized, an efficient lock-based concurrency control protocol is developed, and some NP-hard file allocation problems are found to be solvable in polynomial time. The correctness and the feasibility are demonstrated by simulations using a queueing model. The performance of various control strategies is also evaluated. The statistical results show that DDBLMN is a promising design.

**INDEX TERMS**: Concurrency control, file allocation, local computer network, multiaccess bus, query processing, transaction.

## 1. INTRODUCTION

The steadily increasing demand on the information processing capability of computer systems and the deficiency of conventional computer architectures for database management have led to the design of DDBs, or *distributed database systems*. A DDB is a collection of data that belong logically to the same system, but are spread over multiple computers connected by a network [7]. Some of the key problems in designing DDBs are the distribution of data, the distributed processing of queri the concurrency control of transactions, and the design the supporting communication network.

*Concurrency control* maintains the integrity of inf mation in a multiuser environment, and prevents upda by one user from interfering with retrievals and upda of another. Concurrency control is complicated by need to concurrently access information in multiple co puters. Since communication delays usually prohi instantaneous distribution of status information, lo systems may not have up-to-date global system stat Further, a computer or the network may fail, a recovery must be possible. The processes in the syst should maintain data integrity by cooperately exchang status information through a concurrency control pro col [4,18].

*Distribution design* considers the distribution of da base fragments in the system. To increase the availabi of data and to reduce the overhead of data access, d are usually replicated in more than one site at the cos increased overhead of data storage and update. problem is usually formulated into the FAP, or *allocation problem*, which is to minimize the ove

This research was supported by CIDMAC, a research unit of Purdue University, sponsored by Purdue, Cincinnati Milacron Corporation, Control Data Corporation, Cummins Engine Company, Ransburg Corporation, and TRW.

* Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210. (614) 422 5236.
lien@ohio-state.arpa, {ihnp4,cbosgd}!osu-eddie!lien.

** Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801.
(217) 333 3516. wah%uicsld@uiuc.arpa.

interface functions among all other modules. It also maintains a global directory and provides transparent accesses to users.

(2) **Transaction Handlers** (TH) are created by the Transaction Monitor to handle transactions. There is one TH for each local or remote transaction.

(3) **Data Server** serves as a local database which accesses and updates the local database.

(4) **Concurrency-Control Server** monitors the actions of all local and remote transactions on the network, maintains all necessary status information, and performs concurrency control.

(5) **Network-Interface Server** performs all the interface functions to the communication network and provides guaranteed end-to-end message delivery.

### 2.2. Transaction Processing

The transaction processing model in DDBLMN is similar to System R* [17]. In system R*, a process is created for each transaction and exists for the duration of the transaction session. Hence, there is only one setup cost for each transaction, and keeping track of the sequence of requests within a transaction is simplified.

Referring to the model in Figure 1, the TM of the site at which a transaction originates (called the *home site* of the transaction) initiates a TH to handle the transaction. This TH may then ask the TMs at other sites to initiate THs to process the transaction together. All these THs belong to the same transaction and are retained until the transaction is terminated. In this way, a transaction is carried out by a set of cooperating THs (Figure 2).

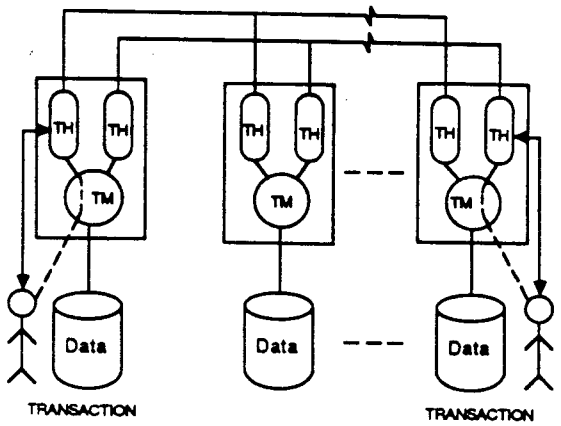



Figure 2 Example of two transactions being processed by DDBLMN.

The Concurrency-Control Server is the process in each site to handle concurrency control. Each access or update request generated in a site is sent to the local Concurrency-Control Server, which will then check its consistency according to the concurrency control protocol. If there is no conflict, the request will be sent to the local Data Server. The processing of a transaction consists of three phases.

(1) **Preprocessing Phase** (*initial processing*): In this phase, a new transaction is processed at the home site. The sites to be involved in the query processing (*related sites*) are identified. Each query in the transaction is restructured into a suitable form for efficient processing.

(2) **Initial Broadcast Phase**: The restructured transaction is broadcast to all sites. The broadcast distributes the transaction and the information for concurrency control (intention list) to all related sites. Initial broadcasts may be given higher priorities to transmit in order to distribute transactions to all sites as early as possible.

(3) **Independent Query Processing Phase**: Each query is processed in sequence by all related sites after the initial broadcast. Strategies for DQP are presented in Section 2.4. The consistency of queries is maintained by the concurrency control algorithm and will be discussed in Section 2.5.

In the remaining sections, we will show how the characteristics of a local broadcast bus can be effectively used to improve the performance of a DDB. The performance study using a queueing network model will also be presented.

### 2.3. File Allocation, Query Processing, and Concurrency Control

In this section, we summarize the results on file allocation, query processing, and concurrency control we developed for DDBLMN. More detailed descriptions can be found elsewhere [22,23,29].

#### File Allocation

It is assumed that file allocation is done at the time of database initialization and that the average access rates to files from each site can be estimated, hence the problem can be independently solved from the query processing and concurrency control problems.

the use of explicit lock massages. Further, the sequential transmission property of the network allows the database to be driven by network events. These two properties allow the concurrency control problem on DDBLMN to resemble that on a centralized system. Therefore, a lock-based protocol is a natural choice since the lock management overheads are significantly reduced, while expensive roll-backs are avoided. Since activities and the intention list of each transaction are known to all sites through the broadcast network (the intention list can be obtained from the initial broadcast), each site maintains a *Broadcast Transaction Table* (BTT) that records the status and intention lists of all transactions in the system as well as the status of referenced relations (locked or unlocked). Based on the BTT, a *Precedence Graph* (PG) recording the precedence relationships among all transactions is also maintained in each site. BTTs and PGs in all sites are always identical since they are driven by the unique network events. When a transaction is processed, only the local BTT has to be checked to determine the serializability condition. A transaction will either wait until serializability is satisfied or change its schedule. Locks on local objects are broadcast to enforce global consistency. A successful broadcast is sufficient to grant the lock to the requesting transaction since only one lock can be broadcast at any time. The global lock broadcast can be embedded in other network activities to further reduce the communication overhead. Note that the precedence order between two transactions is fixed after the time that the first conflicting access of these transactions is detected. Once it has been decided that one transaction will precede another transaction, the preceding transaction will never wait for locks held by other transaction, thereby avoiding deadlocks. More detailed descriptions can be found elsewhere [23].

## 3. PERFORMANCE EVALUATION

The DDBLMN has been simulated on a number of SUN-2 workstations running 4.2BSD UNIX* operating system. The simulations were intended to demonstrate the correctness of the design, to pinpoint the potential problems in implementations, and to compare the performance of various control strategies. They were not intended to evaluate the exact performance of DDBLMN. We have compared the performance between

(1) redundant and non-redundant materialization, and

(2) releasing locks after the LP phase and after the R phase.

Having a second copy of a relation in the working stora allows a transaction to release read locks after the L phase. These strategies were compared by varying

(1) the ratio of the number of update transactions to t total number of transactions,

(2) the number of transactions that a site can handle,

(3) the ratio of packet size in the communication syste to the block size of disks, and

(4) the transaction arrival rate.

### 3.1. Queueing Network Model

Figure 3 depicts the *event-driven* queueing model DDBLMN. The physical resources are modeled as server which include DISKs, CPUs and the NETWORK. TH logical resources (relations) are controlled by the co currency control protocol and cannot be modeled as simple server. However, a pseudo server 'THINK' is use to model the lumped waiting time in transaction proces ing. A request for service, the start of service, and the te mination of service are events. The routing of tokens dependent on the state of the system. An active TH in site is represented as a token, which circulates around th servers in a site and the server NETWORK. A token in

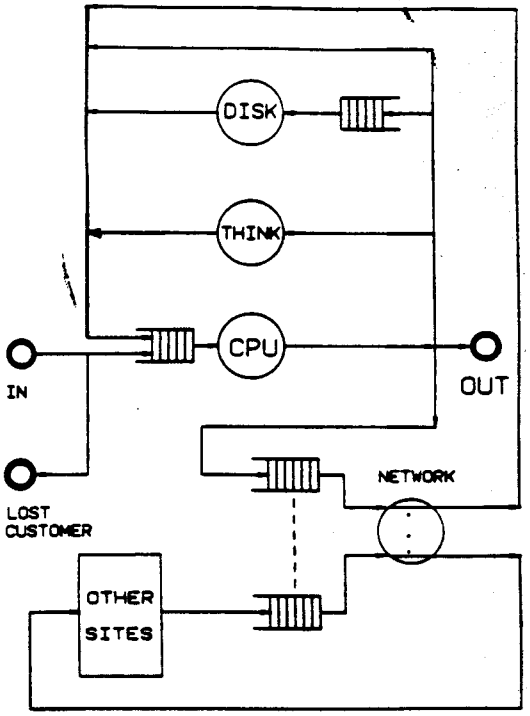



Figure 3 Queuing model of simulator DDBLMN-SIM.

* UNIX is the trademark of AT&T. 4.2BSD is Berkeley version 4.2.

### 3.3. Assumptions in the Simulations

To allow the simulation to be conducted in a reasonable amount of time, the following assumptions were made.

(1) There was one query in each transaction.

(2) There was at most one attribute to be updated.

(3) The attribute to be updated was also a queried attribute, hence a write was always preceded by a read.

(4) When non-redundant materialization was used, a copy of each queried relation was randomly selected.

(5) After semi-joining with an attribute of selectivity p, a relation (say $R_i$) was reduced from the original size of $s_i$ to a size randomly distributed within 30% deviation of the expected size $s_i \cdot p$. The 30% deviation was arbitrarily chosen here. Effects of indirect semi-joins were ignored.

(6) The heuristic function used in the GSJ phase was assumed to be the product of the effective size and the relative selectivity of a candidate minimum attribute.

(7) The join restriction effect in the RT phase was ignored since it was insignificant after a sequence of semi-joins.

(8) The transmission order in the RT phase was governed by the site numbers.

(9) Since only one message can be broadcast from any site at any time, the Network-Interface Server in each site was assumed to process one network access request at any time.

(10) The service time of a CPU server was independent of the number of processes sharing the CPU. This assumption was made because the CPU overhead was about one to two orders of magnitude less than that of disks and network.

### 3.4. Simulator DDBLMN-SIM

A simulator DDBLMN-SIM was written in the C programming language using a event driven simulation management package DDBLMN-SMPL, a modification of SMPL [28]. The simulator was developed on a VAX-11/780 running 4.2BSD UNIX operating system at the Department of Computer and Information Science, the Ohio State University, and was executed on the SUN-2 workstations there.

The parameters used to drive the DDBLMN-SIM a shown in Tables 1 thru 7. Table 1 shows the paramete for the system configuration. Due to time constraints the simulations, only four sites, four relations, and fo joining domains were used. The small number of rel tions with respect to the number of sites may increase t possibility of update conflicts and hence a larger overhe as compared to a system with a larger number of rel tions. The simulation results show that the proposed pr tocol works well even under this situation, thus shou perform satisfactorily when the number of relations increased. We assume that each site can initiate as mar THs as possible. However, the maximum number of tra sactions entering a site is limited. As a result, the numb of THs that a site may generate is also limited. The ca dinalities of the joining domains, the width of the joinir domains, the width of the relations, and the initial card nalities of relations are also shown in Table 1. Th configurations of the relations are shown in Table 2. Th initial selectivity of each joining attribute, which is th ratio of the initial cardinality of the attribute to the ca dinality of the domain, is listed in Table 3. The alloc tion of relations is shown in Table 4. These values we all randomly generated.

Table 1 Configuration of DDBLMN-SIM.

| | |
|---|---|
| No. of sites | 4 |
| MAX_TR_OF_SITE | 5,8,10 |
| No. of relations | 4 |
| No. of joining domains | 4 |
| Cardinalities of joining domains | 80, 80, 160, 200 |
| Width of joining domain | 16, 8, 24, 40 |
| Width of relations | 240, 376, 408, 368 |
| Initial cardinalities of relations | 38707, 40960, 193536, 30720 |

MAX_TR_OF_SITE: max. no. of transactions that ca enter a site at the same time.

Table 2 Configuration of relations.

| | dom 1 | dom 2 | dom 3 | dom 4 |
|---|---|---|---|---|
| Rel 1 | 0 | 0 | 1 | 1 |
| Rel 2 | 0 | 1 | 1 | 0 |
| Rel 3 | 1 | 1 | 1 | 0 |
| Rel 4 | 1 | 0 | 0 | 1 |

Table 8 Tunable variables.

| | |
|---|---|
| TR_INTERARRIVAL_TIME | 0.1, 0.3, 0.5, 0.7 |
| MAX_TR_OF_SITE | 5,8,10 |
| PACKET_SIZE (bytes) | 100, 500, 1000, 2000 |
| UPDATE_QUERY_HIT (or UQH) | 0, 0.4, 0.7, 1 |
| R_LOCK (or r_unlock_time) | 0, 1 |
| RED | 0, 1 |

PACKET_SIZE: packet length in bytes

R_LOCK: the time that a read lock can be released
0: after RT phase
1: Any time after use

RED: materialization strategy
0: non-redundant
1: redundant

in the state transition and the token routing. The system requires each site to keep track of the complete state of the database. To save communication overhead, state transition is computed by each site independently based on the network messages received. The state transition is fairly complicated, and an error in this part may lead to deadlocks and erroneous updates. Piggybacking the necessary information in the broadcast message would reduce its complexity and processing overhead.

It is necessary to compute the transitive closure in maintaining the state of the system. A common sequential algorithm represents the precedence graph as an adjacency matrix [1] and has $O(N^3)$ complexity. This overhead is substantial when the matrix is large, especially when the probability of conflict among transactions is low. In this case, the adjacency matrix is most likely sparse. To reduce this overhead, it is observed that when a new precedence is imposed or a transition is completed, it is only necessary to modify the previously computed transitive closure. The following recursive algorithm is used in DDBLMN-SIM to maintain the precedence graph.

**Algorithm DDBLMN-TRANSITIVE**

Consider the graph as a family of isolated acyclic graphs, with one and only one directed edge between any pair of nodes in the graph. Node $N_i$ represents transaction $TR_i$.

(a) Remove the node and all edges connected to it when the transaction that this node represents terminates.

(b) Whenever a new precedence relationship is imposed on $TR_i$ and $TR_j$ (say, $TR_i$ precedes $TR_j$),

(b.1) connect $N_i$ to $N_j$;

(b.2) connect all preceding nodes of $N_i$ to $N_j$ and all succeeding nodes of $N_j$.

Note that this algorithm may not be better than a dire computation of the transitive closure when the probab ity of conflict is high and the adjacency matrix is dense.

### 3.5.2. Observations and Suggestions

The numerical simulation results are shown in Tab 9. Each combination generated from Table 8 was simu lated. The average, maximum, minimum, and standar deviation of 192 combinations of different MITs on th following observations are listed.

(1) **Mean Response time** is the average response tim of all transactions in a simulation.

(2) **Mean queue length of NETWORK** in a simula tion

$$\left( \frac{\sum_{\text{all sites}} \left( \begin{array}{c}\text{mean queue length of} \\ \text{NETWORK–INTERFACE in the site}\end{array} \right)}{\text{number of sites}} \right)$$

(3) **Mean queue length of DISK** is the average of th mean DISK queue length at all sites, i.e

$$\left( \frac{\sum_{\text{all sites}} \left( \begin{array}{c}\text{mean queue length of} \\ \text{DISK in the site}\end{array} \right)}{\text{number of sites}} \right)$$

(4) **Utilization of NETWORK** is the busy time of th NETWORK over the total time in a simulation.

(5) **Mean utilization of DISK** is the average utiliza tion of DISKs over all sites in a simulation.

(6) **Mean ratio of NETWORK service time to th total transaction time** is the average of

$$\left( \frac{\text{total time serviced by the NETWORK}}{\text{total transaction time}} \right)$$

of each individual transaction in a simulation.

(7) **Mean ratio of DISK service time to the tota transaction time** is the average of

$$\left( \frac{\text{total time serviced by the DISKs}}{(\text{total transaction time})\cdot(\text{number of sites})} \right)$$

of each individual transaction in a simulation.

(8) **Mean ratio of CPU service time to the tota transaction time** is the average value of

$$\left( \frac{\text{total time serviced by the CPUs}}{(\text{total transaction time})\cdot(\text{number of sites})} \right)$$

of each individual transaction in a simulation.

(3) From Figures 6 and 7, we can also see that the UPDATE_QUERY_HIT has no significant effect on the length of either the DISK or NETWORK queues. This is true because we assume that at most one relation is updated and the volume of updated data is usually small. This fact also indicates that the communication overhead for lock management in this system is very low as expected.

(4) Figure 8 compares the utilization of DISKs and NETWORK. It is clear that the NETWORK is heavily utilized, especially when the system load is high. The DISK utilization is as low as 20% even when the NETWORK is almost 100% utilized. This indicates that the NETWORK is the bottleneck of the system.

(5) Figure 9 shows the comparison of the average ratio of the simulated times serviced by CPU, DISKs and NETWORK to the total simulated transaction time. It is clear that a large portion of the transaction processing time was spent on the NETWORK. Only less than 10% of the time was spent on the DISKs and less than 5% on the CPU.

(6) The size of packets in the NETWORK has significant impact on the response time as shown in Figure 10. Packets of fixed sizes were used in each simulation. As there are normally a large number of small packets and the overhead in each packet is small (208 bits), hence the smaller the packets are, the better the system performs. To improve the performance, either the additional capacity in these fixed-sized packets can be used to piggyback useful information, or variable sized packets can be used. These alternatives have not been investigated in this paper.

(7) Two alternatives of read_lock releasing times were simulated. When the UPDATE_QUERY_HIT is 0.0, that is, only read-only queries are made, all relations can be released immediately after access. Therefore, having a second copy in the working space is not beneficial and may even degrade the performance. As the UPDATE_QUERY_HIT is increased, having a second copy improves the performance of the system.

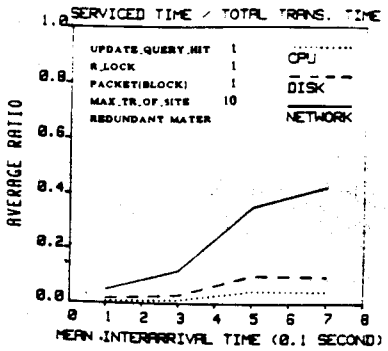



Figure 9 Comparison of the serviced time of CPU's DISK's and NETWORK.

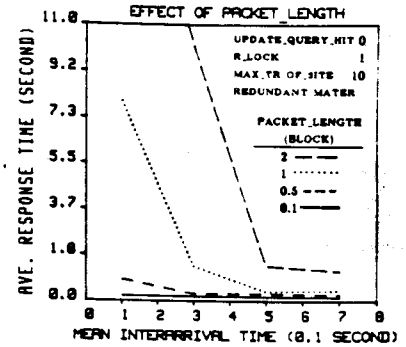


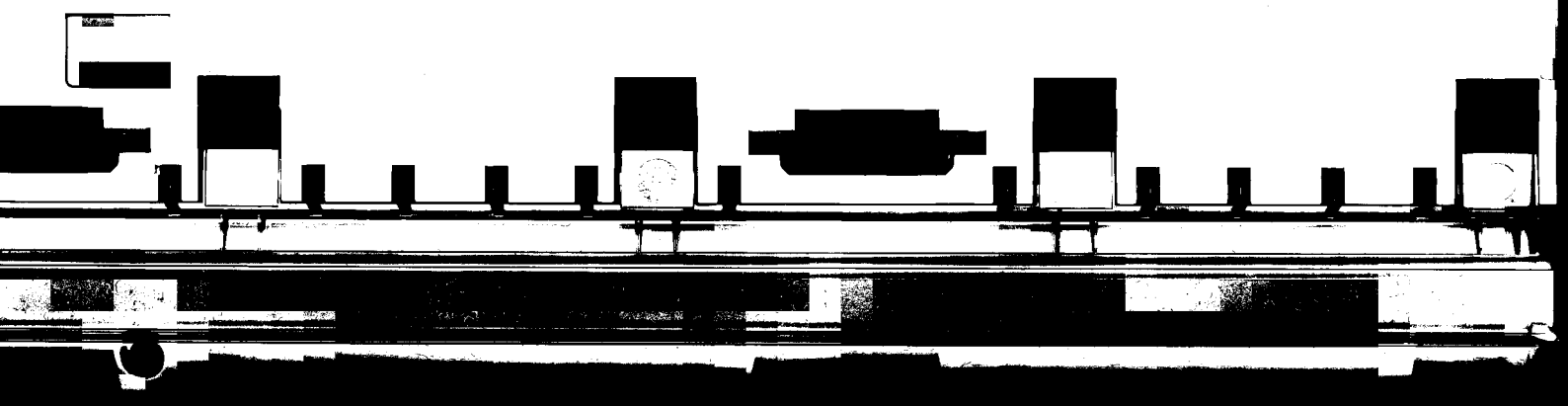

Figure 10 The effect of packet size to the mean response time.

(8) Redundant materialization can improve the perfor mance over non-redundant materialization, but th difference is not significant. This is the case becaus the scale of our simulations was too small such tha at most two copies of a file were allocated. W believe that redundant materialization will be mor attractive when the scale of the system is extended.

We must emphasize that the absolute value in thes simulations may not be close to those of a real systen since the exact parameters are not available. The simula tions are intended to investigate the relative performanc of various design alternatives and to provide guidelines t system designers.

## 4. CONCLUSIONS

In this paper, we have described the design of a dis tributed database on a local computer system connecte by a multiaccess network. A transaction processin model that integrates the control strategies in con currency control and query processing is proposed. Du to the broadcast capability of multiaccess networks, eac update can be processed at a cost independent of th placement and number of copies, and the informatio



operating cost subject to system constraints, such as availability, reliability, and delay. A special case of the FAP is the SFAP, or *simple file allocation problem*, which considers the allocation of multiple copies of a single file, and the effects of queries, updates, and data storage are represented as costs in the system [6,11,13,21].

*Query processing* is to identify and manipulate the portion of database concerned by a query, called *target data*. In processing a multi-site join, either the required files are assembled at a single site and are processed there, or the query and the intermediate results are sent and semi-joined sequentially through a number of sites. The design of an effective query processing strategy to minimize the processing overhead is known as the DQP, or *distributed query processing problem* [2,3,5,10,12,14,15,19,24,25].

Although these problems are interrelated, they are usually studied independently due to the complexity of the combined problem. However, when the network is a multiaccess broadcast bus, the combined problem can be solved easily, and better solutions are obtained. Such is the case in DDBLMN.

In this paper, we present the design and evaluation of DDBLMN, a DDB on a local computer system connected by a reliable *multiaccess* bus with the *broadcast capability* (such as the Ethernet) [23,26]. DDBs that emphasize reliability issues have been studied by Chang [8,9]. The broadcast capability and multi-point configuration are important characteristics for choosing a multiaccess network to support DDBs in a local area environment.

(1) Multiple copies of a piece of data can be updated by one broadcast. This reduces the operating cost and simplifies the design.

(2) The costs of remote accesses and updates are site independent. Again, this simplifies the design.

(3) By monitoring bus activities, global system status information is almost completely available at every site. The overhead for status information exchange is, therefore, very small.

(4) Because of the multi-point configuration and the short propagation delay, information broadcast on the bus is available at every site almost instantaneously.

(5) The multiaccess bus can be used as a gateway for synchronization. When messages are transmitted reliably, all messages arrive at each site in the same order as they are sent [27].

On the other hand, the shared bus is a critical resource because the fraction of bandwidth that each site has is very small as compared to the local processing power. The reduction of the communication overhead is, therefore, one of the critical issues to be addressed.

The data model is assumed to be relational in which a relation is the basic unit to be stored, locked, and queried. A transaction consists of a sequence of queries, each in the form of a conjunctive equijoin with projections and selections. The order of execution of queries in a transaction is assumed to be fixed, but the order of operations within each query is flexible. The directory for locating relations is assumed to be fully replicated at each site.

In Section 2, the system architecture and a transaction processing model are described. The control strategies developed earlier are summarized [22,23,29]. A queueing model to evaluate the performance and the associated simulation results are presented in Section 3.

## 2. SYSTEM OVERVIEW

### 2.1. System Architecture

The logical structure of the local system at each site is shown in Figure 1. Each site consists of five modules: Transaction Monitor, Transaction Handlers, Data Server, Concurrency-Control Server, and Network-Interface Server.

(1) **Transaction Monitor** (TM) coordinates all local and remote transactions by (a) initiating/terminating Transaction Handlers that handle each individual transaction, and (b) providing

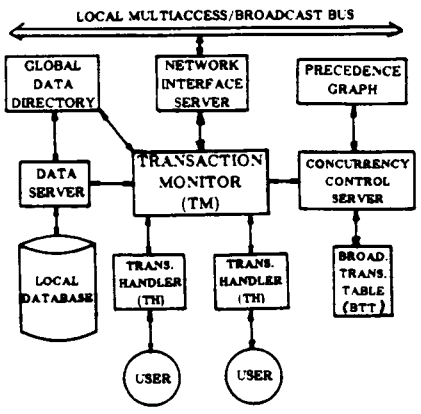



Figure 1 Logical architecture of local sites.

The SFAP can be formulated as an integer programming problem and solved by considering the single-copy and the multicopy cases separately. Optimal allocation in both cases can be solved in O(N) time. The global optimum is obtained by comparing the costs obtained.

The SFAP can also be solved in a similar way when an availability constraint is included. When an average-delay constraint is included, the problem has been proved to be isomorphic to the 0/1 knapsack problem [22,23,29], which can be solved by dynamic programming algorithms or fully polynomial-time approximation schemes. For the general FAP, a branch-and-bound algorithm has been developed to enumerate the allocations of the first two copies of each file and to solve N knapsack problems for each combination. The worst-case number of knapsack problems to be solved is $N^{2m}$, where N is the number of sites and m is the number of files. This is much better than exhaustive enumerations, which has $O(2^{Nm})$ complexity.

In summary, solution algorithms for the file allocation problems are more efficient in the DDBLMN than those in systems with general point-to-point networks.

### Distributed Query Processing

A five-phase dynamic query processing strategy using redundant materialization has been proposed for DDBLMN. A static query processing strategy schedules the sequence of semi-joins before the schedule is executed, while a dynamic query processing strategy schedules the semi-joins according to the current system status. In static processing strategies, the system status in each intermediate step of query processing has be be predicted before the schedule is executed, a difficult task in distributed processing. On the other hand, a dynamic strategy can use the current status information in making a scheduling decision for each query processed and can enforce concurrency control during query processing. The combination of query processing and concurrency control allows relations that are not available to be bypassed, thereby preventing the process from being blocked by locked relations. However, the major obstacle to realize dynamic query processing strategies in a point-to-point network lies in the collection of global system status in real time. This is not a problem in multiaccess broadcast networks, as a single broadcast is adequate to distribute information to all sites. A five-phase dynamic query processing algorithm is proposed here.

**(1) Concurrency Control (CC) Phase**: The locks used in LP phase are requested in this phase. This phase is repeated before each semijoin broadcast and update processing.

**(2) Local-Processing (LP) Phase**: Selections, projections, and local joins are performed in this phase. After local processing, the relations needed by the query in a related site are joined together into a single relation called the *site-relation*. (To avoid the size expansion after local joins, they are postponed to the RT phase in the improved version of the five-phase query processing strategy, since the needed information can be obtained from the original relations.) An attribute in a site-relation is called a *site-attribute*. A slowest-site identification process is initiated at the end of the LP phase to synchronize all related sites and to identify the completion of the LP phase.

**(3) Global Semijoin (GSJ) Phase**: A sequence of semijoins are carried out by a greedy heuristic in this phase. In each iteration, a site-attribute (called the *minimum attribute*) is selected and broadcast to be semijoined with all site-relations at other related sites. The statistics of all site-attributes are collected for the next iteration. A heuristic function that measures the potential of leading to an optimal schedule for each attribute is used in the selection of the joining attributes. The design of this heuristic function is database dependent.

**(4) Relations-Transmission (RT) Phase**: In this phase, the resulting fragments of the site-relations are broadcast sequentially in an arbitrary order to the *post-processing sites*, where the complete join will be performed.

**(5) Post-Processing (PP) Phase**: At the post-processing site, the full join and subsequent operations are executed.

The performance of the proposed query processing strategy relies on the ability of the system to identify the minimum attributes. A distributed extremum identification algorithm with an $O(\log_2 N)$ complexity based on Wah and Junag's algorithm [16,20] was adapted [23,29], where N is the number of related sites.

### Concurrency Control

A simple and efficient lock-based concurrency control protocol was developed in DDBLMN. The broadcast capability of local multiaccess networks allows locks to be known to all sites simultaneously, and hence eliminating

site is generated for an incoming transaction, which in turn generates another set of tokens in other sites after the initial broadcast, each of which represents a TH in a related site. A token is removed either after the transaction terminates or after the site where it is located is no longer involved in the processing. A token is *busy* if it is being served by a physical server (a physical server represents a physical resource); *waiting* if it is waiting for service from a physical server; and *sleeping* otherwise. A token is also called *active* if it is not sleeping.

All processes in one site share the single CPU in a 'process sharing' mode. The service time of a CPU server is proportional to the complexity of the job and the volume of data to be processed. Since the processing time of the local database in each site is dominated by the disk access time, it is modeled as a DISK server working in a FCFS (first-come-first-serve) mode. All sleeping tokens in a site are put in the local server THINK until wakened up by other tokens. There is only one NETWORK server in the model, which operates in a 'random selection' mode to model the behavior of the CSMA/CD protocol. In each site, a FIFO (first-in-first-out) queue is used to store all local tokens (tokens generated from this site) that request NETWORK services. The NETWORK server behaves as a RANDOM server which randomly picks one of the non-empty queues and serves the first arrival in this queue. The service time is also proportional to the volume of data to be transferred.

### 3.2. Process Flow of DDBLMN

The process flow of a transaction is shown in Figure 4. A transaction is first initialized in the home site. After the initial broadcast, all sub-queries are cooperatively processed by a set of related sites. In processing a sub-query, a TH requests read-locks from the local Concurrency-Control Server for local relations used in the LP phase. The Concurrency-Control Server grants the locks to the TH temporarily if conflicts were not detected. At the end of LP phase, a synchronization phase is started to synchronize all sites and to confirm the locks globally. If the synchronization process succeeds, the temporarily granted locks are permanently granted to the transaction. Before these locks are confirmed, any conflicting lock broadcasts on the network will invalidate these temporary locks. The LP phase should be restarted in this case.

Following the successful synchronization, the GSJ phase starts immediately. During this phase, a sequence of

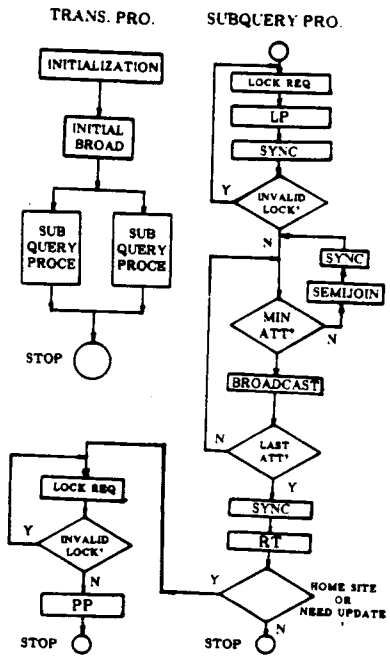



Figure 4 Process-flow diagram of DDBLMN-SIM.

global semi-joins is processed iteratively. In each iteration, a TH requests read-locks for those relations not locked in the LP phase. The minimum attribute is then identified and broadcast to other sites, which execute the semi-joins with the minimum attribute. The GSJ phase is repeated until all the necessary semi-joins are processed. At the completion of the GSJ phase, the RT phase starts. All related sites sequentially broadcast their relations that have not been broadcast by other sites to the home site. A TH not in the home site terminates at the end of the RT phase if there is no relation in this site to be updated. All sites that have relations to be updated continue their processing in the PP phase. The write locks are first obtained in this phase. After the write-locks are temporarily granted by the Concurrency-Control Server, the home site broadcasts the identity of the target data with new values to all sites. The write-locks are confirmed by the broadcast. All sites will update their database and terminate the transaction. The write-locks may be invalidated before the broadcast if a conflicting lock broadcast is detected during this phase. The PP phase should be restarted in this case. All relations to be updated are still read-locked in this phase, hence it is only necessary to restart the PP phase after the conflict has been resolved.

Table 3 Initial selectivities of all joining attributes.

| | att 1 | att 2 | att 3 | att 4 |
|---|---|---|---|---|
| Rel 1 | - | - | 0.6 | 0.9 |
| Rel 2 | - | 0.4 | 0.9 | - |
| Rel 3 | 0.8 | 0.3 | 0.1 | - |
| Rel 4 | 0.6 | - | - | 0.5 |

Table 4 Relation allocation.

| | Site 1 | Site 2 | Site 3 | Site 4 |
|---|---|---|---|---|
| Relation 1 | 1 | 0 | 1 | 0 |
| Relation 2 | 0 | 1 | 1 | 0 |
| Relation 3 | 1 | 0 | 0 | 1 |
| Relation 4 | 0 | 1 | 0 | 1 |

Table 5 Timing parameters (time unit : $10^{-6}$ sec.)

| | |
|---|---|
| CPU | 3.0 * no. of bytes |
| DISK | uniform 0, 20000) + 1000 * no. of blocks |
| BUS | 1.0 * bits/packet * no. of packets |

| | |
|---|---|
| Bus contention slot | 50 |
| Preprocessing time | 1000 |

Table 6 Data format.

| | |
|---|---|
| No. of bits/byte | 8 |
| Disk block length | 1000 bytes |
| Packet length (bytes) | 100, 500, 1000, 2000 |

Timing parameters are shown in Table 5. At a speed of 1 MIPS, we assume that a CPU can process a byte in 3 instruction cycles and that the processing time is proportional to the number of bytes to be processed. The disk access time consists of two parts: the block seek time and the data transfer time. The length of each block is 1000 bytes. The block seek time is a uniformly distributed random variable between 0 and 20 milliseconds, which represents the time to move the disk head and to locate the sector. It is assumed that there is no seek time between adjacent blocks transferred in the same request. Once the first block is located, successive blocks can be transferred continuously with a speed of 1 block per millisecond. The bus communication overhead is represented by a constant plus a variable proportional to the volume of the transferred data. We assume that all necessary initializations are done in the Network-Interface server, and that the contention overhead is accounted for separately. Consequently, the service time of the BUS server is dependent only on the number of transferred packets. The number of overhead bits is assumed to be 208, which is the same as in the Ethernet specification [26]. A contention slot in the CSMA/CD protocol is assumed to be 50 bit-times. The preprocessing phase lasts for 1 millisecond. The data format is listed in Table 6.

In the simulator, there is a single queue of transaction arrivals with an exponentially distributed interarrival time. The mean interarrival time (MIT) varies from 0.1, 0.3, 0.5, to 0.7 seconds. The incoming transactions are randomly routed to a site. A transaction sent to a site is lost if the number of pending transactions at this site has reached the maximum, which is a parameter defined in the simulator.

The query pattern of a transaction is randomly generated. Table 7 shows the parameters used in generating the query pattern. A relation (resp. attribute) is called a *hit relation* (resp. *hit attribute*) if it is involved in the target-data identification in a query. QUERY_REL_HIT (resp. QUERY_ATT_HIT) is the corresponding probability that a given relation is hit by a query. A hit relation must contain at least one hit attribute. In other words, a relation containing at least one hit attribute should be a hit relation. We call a relation containing at least one target attribute a *target relation*. TARGET_REL_HIT (resp. TARGET_ATT_HIT) is the probability that a particular relation (resp. attribute) is a target relation (resp. attribute) in a query. UPDATE_QUERY_HIT is the probability that a query is an update query. A queried relation is randomly selected to be updated if the query is an update. UPDATE_ATT_HIT is the probability that an attribute in a target relation is to be updated. All these probabilities were set to 0.4 except that QUERY_REL_HIT was 0.8 and that UPDATE_QUERY_HIT was varying from 0.0, 0.4, 0.7, to 1.0. All queries were read-only when UPDATE_QUERY_HIT was set to 0.0 and were updates when it was set to 1.0. The values in Table 7 are arbitrarily chosen. Variables that are tunable are summarized in Table 8. Each combination is run with a MIT of 0.1, 0.3, 0.5, and 0.7 seconds.

Table 7 Parameters used for query generation.

| | |
|---|---|
| QUERY_REL_HIT | 0.8 |
| TARGET_REL_HIT | 0.4 |
| QUERY_ATT_HIT | 0.4 |
| TARGET_ATT_HIT | 0.4 |
| UPDATE_QUERY_HIT | 0, 0.4, 0.7, 1 |
| UPDATE_ATT_HIT | 0.4 |

### 3.5. Simulation Results

#### 3.5.1. Implementation Difficulties

The simplicity of the concurrency control algorithms does contribute to the success of the simulator design. However, the major difficulties in the implementation lies

Table 9 Simulation results. (time unit : second)

| Mean Response Time | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 7.053 | 27.900 | 0.190 | 0.5410 |
| 0.3 | 2.739 | 15.900 | 0.160 | 0.2995 |
| 0.5 | 0.833 | 5.190 | 0.170 | 0.0723 |
| 0.7 | 0.482 | 2.080 | 0.140 | 0.0256 |

| Mean(CPU time/total tr. time) | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 0.022 | 0.076 | 0.001 | 0.0018 |
| 0.3 | 0.042 | 0.084 | 0.002 | 0.0020 |
| 0.5 | 0.050 | 0.085 | 0.012 | 0.0016 |
| 0.7 | 0.054 | 0.088 | 0.016 | 0.0015 |

| Utilization of NETWORK | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 0.820 | 1.000 | 0.260 | 0.0200 |
| 0.3 | 0.565 | 0.999 | 0.105 | 0.0247 |
| 0.5 | 0.407 | 0.949 | 0.055 | 0.0209 |
| 0.7 | 0.291 | 0.791 | 0.040 | 0.0160 |

| Mean queue length of NET. | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 4.033 | 8.331 | 0.093 | 0.1851 |
| 0.3 | 1.740 | 6.669 | 0.012 | 0.1560 |
| 0.5 | 0.606 | 3.482 | 0.003 | 0.0665 |
| 0.7 | 0.197 | 1.598 | 0.001 | 0.0201 |

| Mean Utilization of DISK | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 0.259 | 0.453 | 0.064 | 0.0082 |
| 0.3 | 0.128 | 0.185 | 0.064 | 0.0014 |
| 0.5 | 0.084 | 0.104 | 0.064 | 0.0007 |
| 0.7 | 0.057 | 0.080 | 0.045 | 0.0005 |

| Mean queue length of DISK | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 0.108 | 0.368 | 0.002 | 0.0055 |
| 0.3 | 0.019 | 0.067 | 0.002 | 0.0008 |
| 0.5 | 0.006 | 0.021 | 0.002 | 0.0003 |
| 0.7 | 0.002 | 0.006 | 0.001 | 0.0001 |

| Mean(NET. time/total tr. time) | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 0.073 | 0.368 | 0.018 | 0.0038 |
| 0.3 | 0.198 | 0.374 | 0.043 | 0.0071 |
| 0.5 | 0.284 | 0.442 | 0.143 | 0.0068 |
| 0.7 | 0.336 | 0.485 | 0.150 | 0.0078 |

| Lost Trans. (TR. / second) | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 0.221 | 2.081 | 0.000 | 0.0306 |
| 0.3 | 0.000 | 0.000 | 0.000 | 0.0000 |
| 0.5 | 0.000 | 0.000 | 0.000 | 0.0000 |
| 0.7 | 0.000 | 0.000 | 0.000 | 0.0000 |

| Mean(DISK time/total tr. time) | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 0.055 | 0.173 | 0.003 | 0.0044 |
| 0.3 | 0.104 | 0.209 | 0.007 | 0.0051 |
| 0.5 | 0.124 | 0.220 | 0.028 | 0.0043 |
| 0.7 | 0.131 | 0.220 | 0.040 | 0.0040 |

| Total CPU Time (SUN-2) (sec.) | | | | |
|---|---|---|---|---|
| MIT | Ave | Max | Min | Std dev |
| 0.1 | 936.593 | 1914.8 | 464.9 | 21.4398 |
| 0.3 | 732.766 | 1389.6 | 457.9 | 13.9473 |
| 0.5 | 663.789 | 944.4 | 453.0 | 7.6510 |
| 0.7 | 647.708 | 895.7 | 456.0 | 7.1943 |

(9) **Rate of lost transactions per second** is the ratio of the total number of lost transactions to the total simulation time in a simulation.

(10) **Total CPU time** is the total CPU time used in a simulation on a SUN-2 workstation.

The following observations and suggestions can be drawn from the simulation results.

(1) From Figures 5, 6, and 7, it is clear that the system can handle the load when the MIT is longer than 0.5 seconds but will be saturated when the MIT is reduced to 0.1 seconds in many cases. When saturated, the queue of NETWORK is full, and a lot of incoming transactions are lost. This occurs even when no updates are made. When the MIT is longer than 0.3 seconds, the rate of lost transactions is almost nil.

(2) Figure 5 shows that the response time is monotonically increasing with the percentage of updates.

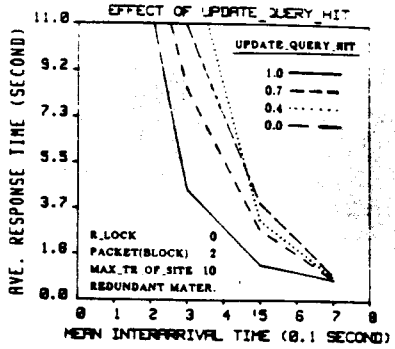



Figure 5 Mean response time.

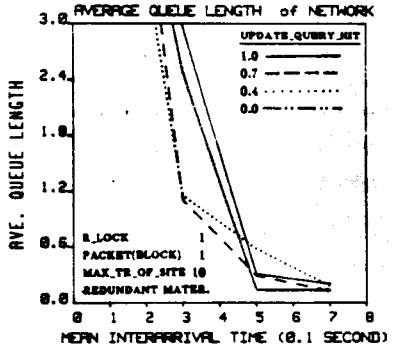



Figure 6 Mean NETWORK queue length.

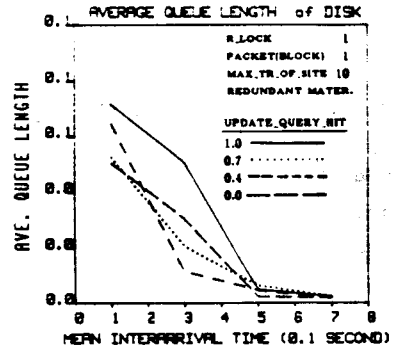



Figure 7 Mean DISK queue length.

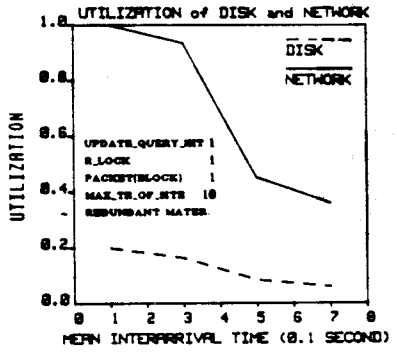



Figure 8 Comparison of the utilization of DISK's and NETWORK.

exchange among all sites is extremely efficient. Based on these characteristics, a dynamic query processing algorithm has been realized, an efficient lock-based concurrency control protocol has been developed, and a number of NP-hard file allocation problems has been found to be solvable in polynomial time when updates are broadcast. The system has been simulated using a queueing model under various conditions and strategies to verify the design, to explore the potential implementation problems, and to project the system performance. Our results show that the proposed approach is feasible and promising. The simulation results provide guidelines to designers in choosing an appropriate control strategy.

## REFERENCES


[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974, pp. 199-200.

[2] P. M. G. Apers, A. R. Hevner, and S. B. Yao, "Optimization Algorithms for Distributed Queries," *IEEE Trans. on Software Engr.*, Vol. SE-9, No. 1, Jan. 1983, pp. 57-68.

[3] P. A. Bernstein and D. M. Chiu, "Using Semi-Joins to Solve Relational Queries," *J. of the ACM*, Vol. 28, No. 1, Jan. 1981, pp. 25-40.

[4] P. A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, Vol. 13, No. 2, June 1981, pp. 185-221.

[5] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie, "Query Processing in a System for Distributed Databases (SDD-1)," *ACM Trans. on Database Systems*, Vol. 6, No. 4, Dec. 1981, pp. 602-625.

[6] S. Ceri, S. Navathe, and G. Wiederhold, "Distribution Design of Logical Database Schemas," *IEEE Trans. on Software Engr.*, Vol. SE-9, No. 4, July 1983, pp. 487-504.

[7] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw-Hill, 1984.

[8] J. S. Chang, "Simplifying Distributed Database Systems Design by Using a Broadcast Network," *Proc. ACM SIGMOD Int'l Conf. on Management of Data*, June 1984, pp. 223-233.

[9] J. S. Chang, "LAMBDA: A Distributed Database System for Local Area Networks," *Database Engineering*, Vol. 8, No. 2, June 1985, pp. 76-83.

[10] D. M. Chiu, P. A. Bernstein, and Y. C. Ho, "Optimizing Chain Queries in a Distributed Database System," *SIAM J. of Computing*, Vol. 13, No. 1, Feb. 1984, pp. 116-134.

[11] W. W. Chu, "Multiple File Allocation in a Multiple Computer System," *IEEE Trans. on Comp.*, Vol. C-18, No. 10, Oct. 1969, pp. 885-889.

[12] W. W. Chu and P. Hurley, "Optimal Query Processing For Distributed Database Systems," *IEEE Trans. on Comp.*, Vol. C-31, No. 9, Sept. 1982, pp. 835-850.

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

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

[15] A. R. Hevner, O. Q. Wu, and S. B. Yao, "Query Optimization on Local Area Networks," *ACM Trans. on Office Information Systems*, Vol. 3, No. 1, Jan. 1985, pp. 35-62.

[16] J. Y. Juang and B. W. Wah, "Unified Window Protocols for Contention Resolution in Local Multiaccess Networks," *Proc. IEEE INFOCOM*, San Francisco, CA, April 1984, PP. 97-104.

[17] B. G. Lindsey, L. M. Haas, C. Mohan, P. F. Wilms, and R. A. Yost, "Computation and Communication in R ," *ACM Trans. On Computer Systems*, Vol. 2, No. 1, Feb. 1984, pp. 24-38.

[18] C. H. Papadimitriou, "The Serializability of Concurrent Database Updates," *J. of the ACM*, Vol. 26, No. 4, Oct. 1979, pp. 631-653.

[19] G. M. Sacco, "Distributed Query Evaluation in Local Area Networks," *Proc. Int'l Conf. on Data Engineering*, Los Angeles, CA, April 1984, pp. 510-516.

[20] B. W. Wah and J. Y. Juang, "An Efficient Protocol for Load Balancing on CSMA/CD Networks," *Proc. 8th Conference on Local Computer Networks*, Minneapolis, MN, Oct. 1983, pp. 55-61.

[21] B. W. Wah, "File Placement on Distributed Computer Systems," *IEEE Computer*, Vol. 17, No. 1, Jan. 1984, pp. 23-32.

[22] B. W. Wah and Y. N. Lien, "The File-Assignment And Query-Processing Problems In Local Multiaccess Networks," *Proc. Int'l Conf. on Data Engineering*, Los Angeles, CA, April 1984, pp. 228-235.

[23] B. W. Wah and Y. N. Lien, "Design of Distributed Databases on Local Computer Systems With A Multiaccess Network," *IEEE Trans. on Software Engr.*, Vol. SE-11, No. 7, July 1985, pp. 606-619.

[24] E. Wong, "Retrieving Dispersed Data From SDD-1: A System for Distributed Databases," *Proc. 2nd Berkeley Workshop on Distributed Data Management and Computer Networks*, May, 1977, pp. 217-235.

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

[26] J. H. Shoch, Y. K. Dalal, D. D. Redell, and R. C. Crane, "Evolution of the Ethernet Local Computer Network," *IEEE Computer*, Vol. 15, No. 8, Aug. 1982, pp. 10-27.

[27] J-S Banino, C. Kaiser, and H. Zimmermann, "Synchronization for Distributed Systems using a Single Broadcast Channel," *Proceedings of the 1st Int'l Conf. On Distributed Computing Systems*, Huntsville, Oct. 1979, pp. 330-338.

[28] M. H. MacDougall, "SMPL - A Simple Portable Simulation Language," *Amdahl Technical Report*, April, 1980.

[29] Yao-Nan Lien, *Distributed Databases On Local Multiaccess Computer Systems*, Ph.D. Dissertation, School of Electrical Engineering, Purdue University, Aug. 1986.