NORTH-HOLLAND

An Evaluation of Network Access Protocols for Distributed Real-Time Database Systems*

Özgür Ulusoy

Department of Computer Engineering and Information Science, Bilkent University, Bilkent, Ankara 06533, Turkey

The results of a considerable number of works addressing various features of real-time database systems (RTDBSs) have recently appeared in the literature. An issue that has not received much attention yet is the performance of the communication network configuration in a distributed RTDBS. In this article, we examine the impact of underlying network architecture on the performance of a distributed RTDBS. In particular, we evaluate the real-time performance of distributed transactions in terms of the fraction of satisfied deadlines under various network access strategies. We also critically examine the common assumption of constant network delay for each communication message exchanged in a distributed RTDBS. © *1997 by Elsevier Science Inc.*

1. INTRODUCTION

A real-time database system (RTDBS) is designed to provide timely response to the transactions of data-intensive applications. Each transaction processed in a RTDBS is associated with a timing constraint typically in the form of a deadline. The research in *distributed* RTDBSs has focused on development and evaluation of new time-cognizant scheduling techniques that can provide good performance in terms of the fraction of satisfied timing constraints. Sha et al. (1991) presented two new real-time concurrency control protocol techniques, called *priority inheritance* and *priority ceiling*, and studied their performance through simulations. Son and Chang (1990) investigated methods to apply the priority-ceiling as a basis for real-time locking protocol in a distributed environment. Some techniques to increase the availability in a partitioned distributed RTDBS were introduced in Lin and Lin (1988). In Ulusoy and Belford (1992), we described several distributed real-time concurrency control protocols and reported the relative performances of the protocols in a nonreplicated database environment. Soparkar et al. (1992) presented an adaptive commit protocol for distributed RTDBS transactions.

In Ulusoy (1994), we investigated the impact of storing multiple copies of data on satisfying the timing constraints of transactions. Various experiments were conducted to observe the performance characteristics of different applications as a function of level of replication. Each application was distinguished by the type and data access distribution of the processed transactions. A detailed performance model of a distributed database system was employed in evaluating the effects of various workload parameters and design alternatives on the system performance. The effects of site failures were also studied to estimate how much replication is needed to provide a reliable processing environment for real-time transactions of different applications.

One interesting question that arises in designing a distributed RTDBS is, "How is the system performance dependent on various characteristics of the communication network connecting data sites?" None of the performance works mentioned above examined the effects of network architectures and protocols on distributed RTDBS performance. The common approach in all those studies was modeling the network as a FIFO server with a fixed service rate independent of the current load and other characteristics of the network.

The effects of networking parameters and communication protocols on "traditional" distributed database systems were investigated by a couple of

*An earlier version of this paper was presented at the First International Workshop on Active and Real-Time Database Systems, Skövde, Sweden, 1995.

Address correspondence to Dr. Özgür Ulusoy, Department of Computer Engineering and Information Science, Bilkent University, Bilkent, Ankara 06533, Turkey.

0164-1212/97/$17.00
PII S0164-1212(96)00043-X

researchers. Sheth et al. (1985) studied the effect of various network parameters on the performance of distributed database systems. They used an analytical model to estimate the delays in communication channels of a long haul network supporting the distributed database system. They showed that the constant transmission time assumption cannot be justified in many cases and that the response time is sensitive to the parameters such as network traffic, network topology, and capacity of communication channels. Özsu and Niu evaluated the effects of network protocols on the performance of some distributed concurrency control algorithms (Özsu and Niu, 1992). Two network protocols, CSMA/CD and token ring, were involved in the evaluations.

In this article, we describe a simulation study of several network access protocols in a distributed RTDBS and address various performance issues. To our knowledge, our work is the first attempt to investigate performance characteristics of the communication network configuration in a distributed RTDBS. Among the questions studied in this work are

- How the performance results obtained with constant network delay assumption are affected when the overhead of message transmission is simulated in detail?
- Which network protocol is the most suited to be used by distributed RTDBSs? What are the basic factors that determine the performance of network protocols in a distributed RTDBS environment?
- Under what conditions is it worthwhile to use real-time network protocols (i.e., protocols that involve timing constraints of communication messages in scheduling their channel access requests)?

The remaining sections are structured as follows. In Section 2, the distributed RTDBS model used in our simulations is presented. Section 3 describes a set of experiments together with our initial findings. It is evaluated in those experiments how the underlying communication network configuration affects the real-time performance of distributed transactions. In Section 4, we conclude our results.

2. MODELING A DISTRIBUTED RTDBS

The performance model is an extension of the model of a distributed RTDBS used in an earlier work of ours (Ulusoy, 1994). The goal of that work was to examine the impact of data replication on the performance of a RTDBS and to analyze the performance trade-offs involved. In that work, we used a data distribution model which provided a partial replication of the distributed database. The model enabled us to execute the system at precisely specified levels of data replication. Each data item was assumed to have N copies in the distributed system, where N can take a value between one and the number of data sites.

Neglecting to model the communication network in detail, in the performance experiments of Ulusoy (1994), it was assumed that the network has enough capacity to carry any number of messages at a given time, and the delay of a communication message between any two data sites is constant. To investigate the issues related to the underlying communication network of a distributed RTDBS, we have extended the system model with a network manager module which accurately simulates the behavior of communication messages exchanged among data sites. The physical structure of the RTDBS model is shown in Figure 1. It is composed of a number of data sites interconnected by a local communication network. Each data site contains a transaction generator, a transaction manager, a resource manager, a scheduler, a buffer manager, and a recovery manager.

The transaction generator is responsible for generating the workload for each data site. The arrivals at a data site are assumed to be independent of the arrivals at the other sites. Each transaction is characterized by a *criticalness* and a *deadline*. The criticalness of a transaction is an indication of its level of importance (Biyabani et al., 1988). It is assumed that each transaction is associated with one of m possible levels of criticalness. The most critical transactions are assigned the highest level. Assignment of criticalness to a new transaction follows a uniform distribution; i.e., the criticalness of the transaction is chosen randomly from the set $\{1, 2, \ldots, m\}$. The deadline of a transaction specifies a certain time in the future the transaction has to be completed before. The transaction deadlines are *firm*; i.e., transactions that miss their deadlines are aborted and disappear from the system. Criticalness and deadline are two independent characteristics of RTDB transactions (Huang et al., 1989; Haritsa et al., 1991). A close deadline does not necessarily imply more criticalness. The transaction manager at the originating site of a transaction T assigns a real-time priority to transaction T based on its criticalness (C_T), deadline (D_T), and arrival time (A_T). The priority of transaction T is determined by the following

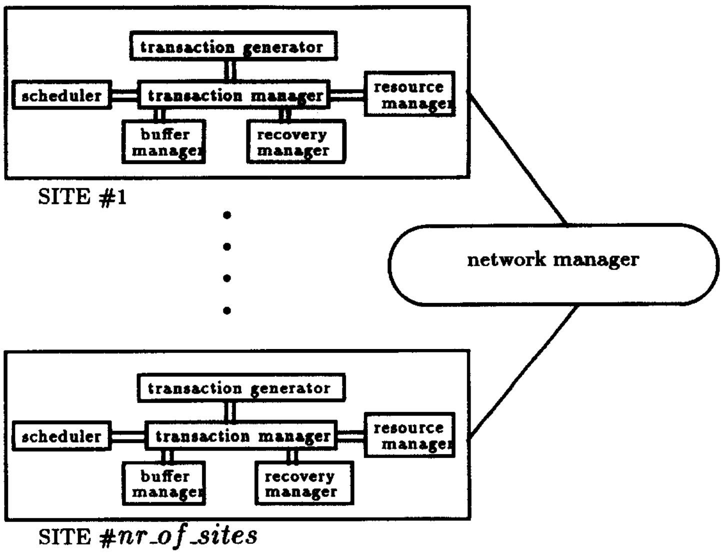

Figure 1. Distributed RTDBS structure.

formula

$$P_T = \frac{C_T}{D_T - A_T}.$$

The priority formula gives equal weight to criticalness and relative deadline. If any two transactions originating from the same site carry the same priority, any scheduling decision between those transactions favors the more critical one; if the transactions are of the same criticalness as well, the transaction with closer deadline is scheduled first. To guarantee the global uniqueness of the priorities, the id of the originating site is appended to the priority of each transaction.

Each distributed transaction exists in the system in the form of a master process that executes at the originating site of the transaction and a number of cohorts that execute at various sites where the copies of required data items reside. A cohort can be defined as a process that performs operations of its transaction on data items stored at a remote site. The transaction can have at most one cohort at each data site. The transaction manager is responsible for creating a master process for each new transaction and specifying the appropriate sites for the execution of the cohort processes of the transaction. The operations of a transaction are executed in a sequential manner, one at a time. For each operation executed, a global data dictionary is referred to find out the locations of the data item referenced by the operation. Each data site is assumed to have a copy of the global data dictionary. After determining which data sites should be accessed for the operation, a cohort process at each of those sites is initiated (if it does not exist already) by the master process to perform the operation in the name of the transaction. Previously created cohorts at those sites are just activated to perform the operation. After the successful completion of an operation, the next operation in sequence is executed by the appropriate cohort(s). When the last operation is completed, the transaction can be committed. The priority of a transaction is carried by all of the cohorts of the transaction.

The effects of a distributed transaction on the data must be made visible at all sites in an *all or nothing* fashion. The so called *atomic commitment* property can be provided by a commit protocol which coordinates the cohorts such that either all of them or none of them commit. It is also necessary in a distributed database system to ensure that *mutual consistency* of the replicated data is provided; in other words, replicated copies must behave like a single copy. This is possible by preventing conflicting accesses on the different copies of the same data item and by making sure that all data sites eventually receive all updates (Garcia-Molina and Abbott, 1987). In our model, the atomic commitment of distributed transactions is provided by the centralized two-phase commit protocol (Bernstein et al., 1987), while the mutual consistency of replicated data is achieved by using the *read-one, write-all-available* scheme (Bernstein and Goodman, 1984).

Access requests for data items are ordered by the scheduler on the basis of the concurrency control protocol executed. An access request of a cohort may result in blocking or abort of the cohort due to a data conflict with other cohorts executed concurrently. The scheduler at each site is responsible for effecting aborts, when necessary, of the cohorts executing at its site. When a cohort completes its data access and processing requirements, it waits for the master process to initiate two-phase commit. The master process commits a transaction only if all the cohort processes of the transaction run to completion successfully, otherwise it aborts and later restarts the transaction. A restarted transaction accesses the same data items as before and is executed with its original priority.

IO and CPU services at each site are provided by the resource manager. IO service is required for reading or updating data items, while CPU service is necessary for processing data items and communication messages. Both CPU and IO queues are organized on the basis of real-time priorities, and preemptive-resume priority scheduling is used by the CPU at each site. The CPU can be released by a cohort process either due to a preemption, when the process commits or it is blocked/aborted due to a data conflict, or when it needs an IO or communication service. Communication messages are given higher priority at the CPU than data processing requests.

Local deadlocks are detected by maintaining a local Wait-For Graph (WFG) at each site. Local deadlock detection is performed by the scheduler each time an edge is added to the graph (i.e., when a cohort is blocked). For the detection of global deadlocks a global WFG is used which is constructed by merging local WFGs. One of the sites is employed for periodic detection of global deadlocks. A deadlock is recovered from by selecting the lowest priority cohort in the deadlock cycle as a victim to be aborted. The master process of the victim cohort is notified to abort and later restart the whole transaction.

Table 1 provides the set of parameters used in specifying the configuration and workload of the distributed RTDBS. The communication network parameters, not listed in this table, will be discussed in the next section. Each data item has exactly N copies in the distributed system, where $1 \leq N \leq$ *nr_of_sites*. Each data site can have at most one copy of a data item. The remote copies of a data item are uniformly distributed over the remote data sites; in other words, the remote sites for the copies of a data item are chosen randomly.

Slack_factor is the parameter used in assigning deadlines to new transactions. The slack time of a transaction is chosen randomly from an exponential distribution with a mean of *slack_factor* times the estimated processing time of the transaction. While the transaction generator uses the estimation of transaction processing times in assigning deadlines, we assume that the system itself lacks the knowledge of processing time information. The deadline of a transaction T is determined by the following formula

$$D_T = A_T + PE_T + S_T$$

where

$$S_T = expon(slack\_factor * PE_T).$$

A_T, PE_T, and S_T denote the arrival time, processing time estimate, and slack time of transaction T, respectively. The formula used to determine the processing time estimate of a transaction in an unloaded system is provided in Ulusoy (1994).

Table 1. Distributed RTDBS Model Parameters

	Configuration Parameters
nr_of_sites	number of data sites
local_db_size	database size originated at each site
N	number of copies of each data item
mem_size	size of the memory buffers used to hold data items at each site
cpu_rate	instruction rate of CPU at each site (MIPS)
instr_process_item	number of instructions to process each data item
disk_access_time	average disk seek + transfer time of a data item (msec)
pri_assign_cost	CPU cost of priority assignment (instructions)
lookup_cost	CPU cost of locating a data item (instructions)
	Transaction Parameters
iat	mean transaction interarrival time at a site
tr_type_prob	fraction of update type transactions
tr_length	mean number of data items accessed by a transaction
data_update_prob	fraction of updated data items by an update transaction
slack_factor	average slack-time/processing-time for a transaction

2.1. The Communication System

There is no globally shared memory in the system, and all sites communicate via message exchanges over the communication network. The network manager is responsible for the transmission of messages among data sites. The message switching component of a data site is called a *node*.

The assumptions of our communication system model are

Table 2. Communication Network Parameters.

Communication Parameters	
nw_bandwidth	network bandwidth (Mbps)
mes_size	message size (bytes)
instr_init_mes	CPU cost to initialize sending/receiving a message (instructions)
instr_per_mes_byte	CPU cost of sending/receiving each byte of a message (instructions)

- The size of the buffers used to hold messages at nodes are infinite; thus, no message loss is experienced due to buffer overflows.
- The communication network is error-free. Therefore, there is no loss of messages and no retransmission is required. Issues such as reliability and fault recovery in communication systems are beyond the scope of this article.

Table 2 lists the communication parameters of the distributed RTDBS model. The parameter *nw_bandwidth* specifies the speed of the network; i.e., the number of bits that can be transmitted per second. *Mes_size* is the length of each message exchanged between the nodes. Each message is processed at its source site prior to its transmission and at its destination site after being received. The message processing overhead, in terms of the number of CPU instructions is simulated using the parameters *instr_init_mes* and *instr_per_mes_byte*. The first of these two parameters corresponds to the initialization cost of transmitting or receiving each message. The second parameter specifies the processing cost of each byte of a message at the source or destination site.

The average CPU delay and network delay experienced by each message can be estimated by using the communication parameters

$$\begin{aligned} &\mathit{CPU\_delay} \\ &\quad = 2 * \frac{1}{\mathit{cpu\_rate}} (\mathit{instr\_init\_mes} + \mathit{mes\_size} \\ &\quad * \mathit{instr\_per\_mes\_byte}) \end{aligned} \tag{1}$$

$$\mathit{network\_delay} = \frac{1}{\mathit{nw\_bandwidth}} * 8 * \mathit{mes\_size}. \tag{2}$$

CPU_delay corresponds to the total processing cost of a message (i.e., sum of the processing costs at both its source site and destination site).

There exist different types of communication messages exchanged to control the execution of a transaction. The message types generated for a particular transaction T are described in Table 3. In the table, the source and destination of each message type are specified using the following notation

TM(S): The transaction manager at site S.
MP(T): Master process of transaction T.
ES(C): Execution site of cohort C.

The discussion of the message types specific to various concurrency control protocols employed in performance experiments is deferred to Section 3.1 which provides the performance results obtained with different concurrency control protocols.

Two different network architecture types are considered in our work: carrier-sense multiple access networks and token ring networks. The Carrier-Sense Multiple Access with Collision Detection (CSMA/CD) is the first network access protocol we explored. In a multiple access network, messages are transmitted on a shared communication channel. Only one message can be successfully transmitted over the channel at any time. In carrier-sense networks, each node that wants to transmit a message should first listen to the communication channel. If any transmission is in progress, the node defers its transmission until the end of the current transmission. Collisions can occur due to the nonzero propagation delay of the communication channel. CSMA/CD protocol provides detection of message collisions. Upon detection of a collision, transmission is aborted and the node schedules its message for the retransmission. The time period over which the node schedules retransmission is doubled each time the message experiences a collision (Bux, 1981).

Table 3. Message Types Generated for Transaction *T*

Message type	Source	Destination	Function
initiate_cohort	MP(T)	TM(ES(C))	To initiate the execution of cohort C of transaction T.
activate_operation	MP(T)	TM(ES(C))	To activate an operation of cohort C.
operation_complete	TM(ES(C))	MP(T)	To indicate that the current operation of cohort C has been completed.
vote_request	MP(T)	TM(ES(C))	To initiate the two-phase commit protocol for T.
participant_decision	TM(ES(C))	MP(T)	To reply the *vote_request* message. The message carries the commit/abort decision of a cohort site.
final_decision	MP(T)	TM(ES(C))	To indicate the final (commit/abort) decision for the commitment of T.

Table 4. Parameters Specific to the Carrier-Sense Multiple Access Network Model

CSMA / CD Parameters	
csma_prop_delay	end-to-end propagation delay
csma_channel_length	length of the communication channel (bits)

Table 5. Parameters Specific to the Token Ring Network Model

Token Ring Parameters	
ring_prop_delay	node-to-node propagation delay
node_latency	delay at each node
ring_length	total length of the ring (bits)

The parameters specific to our CSMA/CD network model are provided in Table 4. The model assumes that time is slotted and nodes can only start transmitting messages at the beginning of each slot. *Csma_prop_delay* denotes end-to-end propagation delay of the communication channel. The parameter *csma_channel_length* specifies the length of the channel in bits (i.e., the maximum number of bits being transmitted on the channel at any instant). The length of a slot is considered to be equal to *csma_prop_delay*; thus, a transmission at the beginning of a slot is recognized by all nodes prior to the next slot. A collision can occur only between the messages that are transmitted at the same slot.

Token ring is the other network access protocol adapted to our communication system. In a token ring, access to the communication channel is controlled by passing a special frame, called token, around the ring. When no message is in transmission, a free token circulates around the ring. When a node becomes ready to transmit a message, it changes the token to busy and puts its message onto the ring. The sending node is responsible for removing its own message from the ring. At the end of its transmission, the node passes the access permission to the node down stream by generating a new free token. Because there is only one token on the ring at any time, there is no contention among the nodes to access the ring (Bux, 1981).

Table 5 describes the additional communication parameters for the token ring model. *Ring_prop_delay* specifies the propagation delay of messages from one node to another. It is assumed that all nodes are equally distanced on the ring. Each message is passed from one node to another on its path from source site to destination site. Each node passes the message on after a short delay, which is specified by parameter *node_latency*. The token circulates around the ring in a time equal to the sum of propagation delays between nodes plus the sum of node latencies.

3. SIMULATION EXPERIMENTS

The simulation program, capturing the details of the distributed RTDBS model, was written in CSIM (Schwetman, 1986), which is a process-oriented simulation language based on the C programming language.

Table 6 presents the default parameter values used in each of the experiments. All sites of the system were assumed identical and operating under the same parameter values. It was assumed that one CPU and one disk unit exist at each data site. The settings used for configuration and transaction parameters were basicly taken from our earlier experiments (Ulusoy, 1994). It was intended by those settings to execute the transactions under high levels of data contention. The default values used for the communication parameters can be accepted as reasonable approximations of what can be expected from today's local communication networks. The value of *csma_prop_delay* is determined as follows

$$csma\_prop\_delay = \frac{csma\_channel\_length}{nw\_bandwidth}$$
$$= 5*10^{-3} msec.$$

Table 6. Performance Model Parameter Values

Configuration Parameters	
nr_of_sites	10
local_db_size	200 data items
N	5
mem_size	500
cpu_rate	20 MIPS
instr_process_item	160000 instructions
disk_access_time	18 msec
pri_assign_cost	20000 instructions
lookup_cost	20000 instructions
Transaction Parameters	
iat	400 msec (exponential)
tr_type_prob	.5
tr_length	6
data_update_prob	.5
slack_factor	5 (exponential)
Communication Parameters	
nw_bandwidth	10 Mbps
mes_size	512 bytes
instr_init_mes	20000 instructions
instr_per_mes_byte	3 instructions
csma_channel_length	50 bits
node_latency	$0.5*10^{-3}$ msec
ring_length	50 bits

Similarly, the value of *ring_prop_delay* can also be found using the other network parameter values.

$$ring\_prop\_delay = \frac{ring\_length}{nr\_of\_sites * nw\_bandwidth} = 0.5 * 10^{-3} msec.$$

The performance metric we used, i.e., *success_ratio*, combines the performance measurements of all criticalness levels, in terms of the fraction of satisfied deadlines, using a specific weight for each level. This metric is defined as follows

$$success\_ratio = \frac{\sum_{i=1}^{m} w_i * success\_ratio_i}{\sum_{i=1}^{m} w_i},$$

where

i: Criticalness level.
m: Total number of criticalness levels ($m = 3$ in our simulations).
w_i: Weight of criticalness level i.
$success\_ratio_i$: Fraction of satisfied deadlines for the transactions of criticalness level i.

The determination of the weights of criticalness levels is highly dependent on the particular application environment (Biyabani et al., 1988). We used linearly increasing weights; i.e.,

$$w_i = i, \quad (i = 1, 2, \ldots, m).$$

For each experiment, the final results were evaluated as averages over 25 independent runs. Each run continued until 1000 transactions were executed at each data site. Ninety percent confidence intervals were obtained for the performance results. The width of the confidence interval of each data point is within 4% of the point estimate. In displayed graphs, only the mean values of the performance results are plotted.

3.1. Evaluation of Concurrency Control Protocols

In Ulusoy (1994), we evaluated the performance of a number of RTDBS concurrency control protocols under different levels of transaction load. The protocols were different in the way real-time priorities of transactions are involved in scheduling data access requests. Concurrency control protocols that employ restarts in resolving conflicts (e.g., optimistic protocols), exhibited better performance than the protocols that use blocking (e.g., locking protocols) when the system was lightly loaded (i.e., for large *iat* values). With optimistic protocols, there is no overhead of transaction blocking due to data conflicts until commit time. Because the number of conflicts is small under low load levels, only a few transactions fail to be validated at commit time. On the other hand, when the transaction load was high, the performance of restart-based protocols was worse compared to blocking-based ones. The overhead of executing a concurrency control protocol that uses restarts in resolving conflicts was observed to be higher than that of a blocking-based protocol due to the large number of restarts experienced under high levels of system load.

The same experiment is repeated here to see how the results obtained are affected when the transmission of communication messages are implemented in full detail. We categorize the concurrency control protocols into two classes as locking protocols that use blocking in resolving conflicts and optimistic protocols that are based on restarting. This section provides the results for one protocol from each class chosen as representative. We first provide a brief description of each protocol together with the summary of the performance results obtained with the constant message transmission and service times assumption.

Priority Inheritance protocol (PI). The priority inheritance method, proposed in Sha et al. (1991), ensures that when a transaction blocks higher priority transactions, it is executed at the highest priority of the blocked transactions; in other words, it inherits the highest priority. The aim is to reduce the blocking times of high priority transactions.

Optimistic Wait-50 protocol (OPT). OPT is an optimistic concurrency control protocol incorporating real-time priorities of transactions (Haritsa et al., 1990). The validation check for a committing transaction is performed against the executing transactions and if the write-set of the validating transaction intersects with the read-set of one of the executing transactions, these two transactions are said to be in conflict. The proposed protocol uses a *50%* rule as follows. If half or more of the transactions conflicting with a committing transaction are of higher priority, the transaction is made to wait for the high priority transactions to complete; otherwise, it is allowed to commit while the conflicting transactions are aborted. While the transaction is waiting, it is possible that it will be restarted due to the commit of one of the conflicting transactions with higher priority. The validation check for a transaction is performed at each data site where a cohort of the transaction has been executed.

The concurrency control protocols were found to be somewhat different in their sensitivity to the constant message overhead assumption. Table 7 pro-

Table 7. Improvement in *success_ratio* by PI over OPT.

↓protocol, *iat* →	300	340	380	420	460
No access protocol	17%	11%	5%	−1%	−2%
CSMA / CD	12%	7%	3%	−2%	−3%
Token Ring	16%	10%	7%	−1%	−3%

Improvement is shown under varying average transaction interarrival time *iat* (in msec) with the constant message overhead assumption, the network access protocol CSMA/CD, and the token ring.

vides the improvement in *success_ratio* obtained with concurrency control protocol PI over protocol OPT under various network access strategies.[1] The line indexed by "No access protocol" provides the evaluation results obtained without employing a specific network access protocol in transmitting messages.[2] When the token ring protocol was employed, the comparative performance results of PI and OPT under different system loads were not much different from those obtained without implementing the details of a network protocol. On the other hand, when CSMA/CD was employed, the performance improvement provided by PI over OPT under high transaction loads was at a lower level. This result might be due to larger number of communication messages involved in implementing the concurrency control protocol PI. The protocol requires that whenever a cohort of a transaction inherits a priority, the scheduler at the cohort's site notifies the transaction's master process by sending a priority inheritance message which contains the inherited priority. The master process then propagates this message to the sites of other cohorts that belong to the same transaction, so that the priority of the cohorts can be adjusted.[3] With protocol OPT, on the other hand, no extra messages are involved for concurrency control because the information necessary for the validation of a transaction is piggybacked on the messages of the two-phase commit protocol. The larger number of messages issued with PI affects the comparative performance of protocols when CSMA/CD is employed. The degradation in the performance of protocol PI can be explained by the waste of time experienced due to message collisions with CSMA/CD. The number of collisions increases as more messages contend for channel access.

Figure 2 displays the real-time performance results of concurrency control protocols PI and OPT with network access protocols CSMA/CD and token ring. For low levels of transaction load (i.e., large *iat* values), CSMA/CD leads to slightly better performance for both PI and OPT. The worse performance of token ring can be due to the delay experienced by ready messages while waiting for a free token. Comparing the concurrency control protocols under high loads, it can be seen that OPT cannot reach the real-time performance level achieved by PI under any network access protocol. The reason for this result, as we explained before, is the waste of resources experienced with OPT due to restarting failed transactions at the end of their executions.

3.2. Evaluation of Real-Time Network Access Protocols

In this section, we provide an investigation of the performance impact of employing priority-based network access protocols in a distributed RTDBS. Each message transmitted carries the priority which is associated with its transaction. The real-time network access protocols selected for evaluation are: the virtual time CSMA/CD protocol (VTCSMA/CD) (Zhao and Ramamritham, 1987), and the IEEE 802.5 Token Ring protocol (Token Ring Access Method, IEEE 802.5 Local Area Network Standard, 1985).

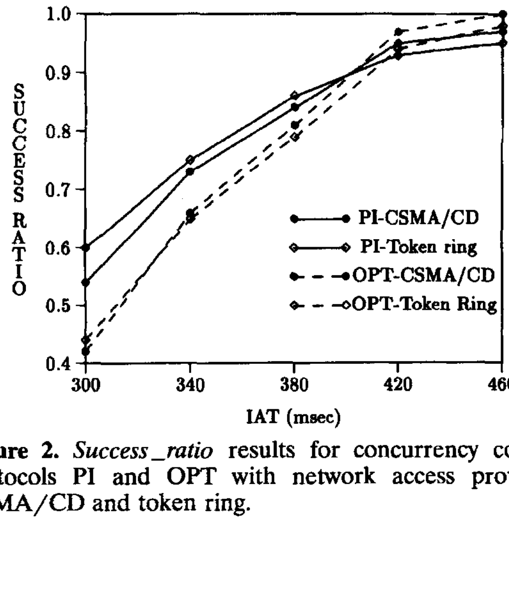

Figure 2. *Success_ratio* results for concurrency control protocols PI and OPT with network access protocols CSMA/CD and token ring.

[1]The range (300 msec, 460 msec) of *iat* values used in the experiments corresponds to an expected CPU utilization of about .90 to .59 at each data site (Ulusoy, 1992).

[2]In those evaluations, the constant values used to simulate the delay of a communication message between any two sites and the CPU time to process a communication message were determined using Equations (1) and (2).

[3]The other locking protocols also require exchange of various kinds of control messages between sites during the execution of a transaction.

Virtual Time CSMA / CD Protocol. The virtual time CSMA/CD protocol (VTCSMA/CD) was proposed by Zhao and Ramamritham (1987) for real-time communication systems. In this protocol, each node maintains two clocks: a real time clock and a virtual time clock. Whenever a node finds the channel to be idle, it resets its virtual clock. The message with the minimum virtual time to start transmission (*VS*) is transmitted first. Transmission begins when the virtual clock equals the *VS* of the message. The virtual clock stops running when transmission begins and starts running (after resetting its value to the time on the real clock) when the channel is idle following completion of transmission or a collision. It runs faster than the real clock.[4] In our experiments, we set the *VS* of a message to the deadline of its transaction.

IEEE 802.5 Token Ring Protocol. In this protocol, the token contains a priority field and a reservation field. A node that has a ready message has to wait until it captures the free token with a priority less than or equal to its priority. The node can try to reserve the next token by writing its message priority into the token's reservation field. However, if a higher priority has already been claimed in the reservation field, the node is not allowed to update it. Following a message transmission, the sender node generates a free token with the priority that has been reserved, if any; otherwise, the priority field of the free token is set at the present priority level.

The VTCSMA/CD protocol has the implementation overhead of delaying the transmission of a ready message until the *VS* of the message becomes equal to the virtual clock. Implementing the IEEE 802.5 token ring protocol, on the other hand, involves an extra processing cost due to comparing the priority of a ready message against the priority field or the reservation field of the token, and setting those fields whenever the conditions hold.[5]

The first experiment investigated the performance of the real-time network access protocols for varying transaction loads (and thus varying message loads). The *iat* parameter was varied from 300 to 460 mseconds in steps of 40. PI was the concurrency control protocol used in the experiments (similar performance characteristics were observed for protocol OPT).

In Figures 3 and 4, the performance results are compared to those obtained with protocols CSMA/CD and token ring which do not involve real-time priorities in scheduling the transmission of messages. Although both real-time network access protocols were observed to provide an improvement over the performance of their nonreal-time counterparts under high levels of transaction load, the improvement provided by VTCSMA/CD over CSMA/CD was not significant. The channel access delay experienced due to the implementation of a virtual clock prevents protocol VTCSMA/CD to become more effective in terms of the real-time performance. Under low levels of transaction load, the real-time network access protocols perform worse than their nonreal-time counterparts. This result shows that when the number of messages contending for channel access is small (as a result of low transaction load), the performance advantage gained by the real-time protocols is outweighed by their implementation overhead. In conclusion, if the system is characterized by low transaction load, it is not worthwhile to use a network access protocol that exploits the real-time priorities.

In another experiment, it was evaluated how successful the transactions are in satisfying their deadlines under different levels of data replication. In conducting data replication experiments, we considered two different application environments, each characterized by the fraction of update transactions processed. The majority of the transactions in the first application are read-only (update transaction percentage: 25%), while the second application is dominated by update transactions (update transac-

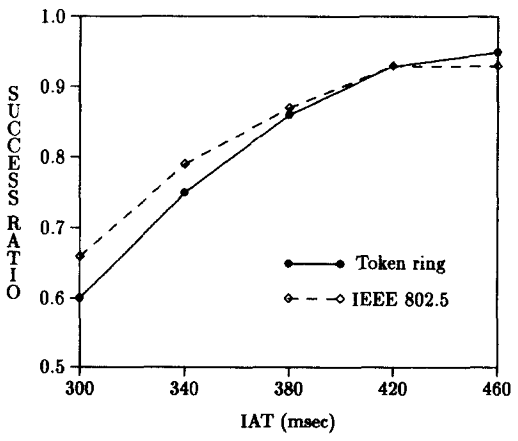

Figure 3. *Success_ratio* results for network access protocols token ring and IEEE 802.5.

[4]Zhao and Ramamritham (1987) provides experimentally the best values for the rate at which the virtual clock runs under different loading conditions.

[5]In our experiments, this extra cost is simulated explicitly by doubling the value of *node_latency* each time a node needs to check or set the priority/reservation fields of the token.

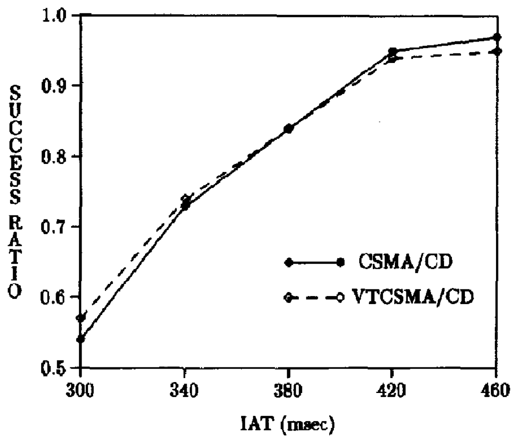

Figure 4. *Success_ratio* results for network access protocols CSMA/CD and VTCSMA/CD.

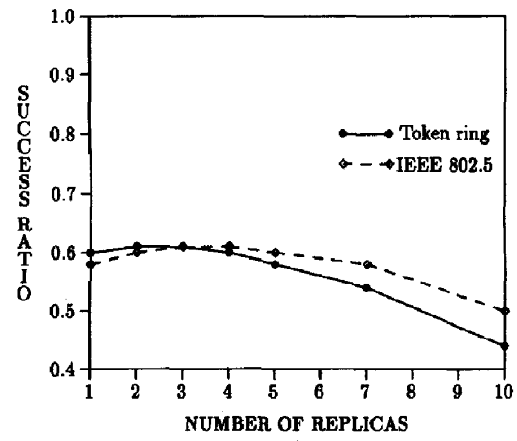

Figure 6. *Success_ratio* vs *N* (number of data replicas) for network access protocols token ring and IEEE 802.5 in an execution environment where update transactions predominate.

tion percentage: 75%). In evaluating the effects of level of data replication on system performance, the number of replicas of each data item (N) was varied from 1 to *nr_of_sites* (*nr_of_sites* = 10). Remember that the consistency of replicated data is provided through the *read-one, write-all-available* scheme. A read operation requires a remote access if a copy of the required data item does not reside locally. In this experiment, the mean interarrival time value (*iat*) was fixed at 400 msec.

The comparative performance results of network access protocols token ring and IEEE 802.5 are displayed in Figures 5 and 6 for two different application environments. With the first application environment, where read-only transactions predominate,

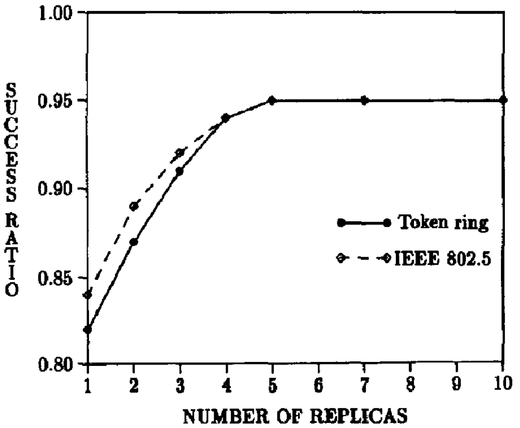

Figure 5. *Success_ratio* vs *N* (number of data replicas) for network access protocols token ring and IEEE 802.5 in an execution environment where read-only transactions predominate.

the fraction of satisfied deadlines is at a higher level (Figure 5) compared to the other application environment where the majority of transactions are of update type (Figure 6). The number of conflicts among the transactions increases when the fraction of update operations becomes higher, which results in a degradation in the performance of the RTDBS. In Figure 5, with both protocols token ring and IEEE 802.5, an improvement in the performance is observed up to a certain point by increasing the data replication level. This improvement is due to the increasing number of local read operations that leads to a decrease in network traffic. After a certain number of replicas, further improvement is not possible because the overhead of multiple copy updates (although they are infrequent) outweighs the performance benefits of the local read operations. With the query-oriented application environment, IEEE 802.5 protocol provides better performance than the conventional token ring protocol when the level of data replication is low. This shows that, in an execution environment where most of the transaction operations require remote accesses, it is advantageous to make use of real-time priorities of communication messages in scheduling their accesses to the communication channel.

Figure 6 provides the real-time performance results for the application environment where most of the transactions are of update type. A considerable degradation in performance is observed if the level of data replication is increased beyond 3. The overhead of update synchronization among the multiple copies of updated data increases with each additional data copy. More communication messages need to be exchanged among sites to provide update synchronization. It is evident from the comparative performance results displayed for protocols token ring and IEEE 802.5 that involving real-time priorities in scheduling network accesses reduces the steep degradation in real-time performance which is experienced as the number of data copies (and thus the number of communication messages) increases.

When the experiment was repeated with the carrier-sense network access protocols, the results obtained for the comparative performance of protocols CSMA/CD and VTCSMA/CD were qualitatively in agreement with the results of token ring and IEEE 802.5. However, it was observed that, under an update-dominant execution environment, data replication has more crucial effects on the real-time performance with the carrier-sense protocols CSMA/CD and VTCSMA/CD. The drop in *success_ratio* as a result of increasing the level of replication beyond a few is more steep (see Figure 7) compared to the results of ring protocols and the results obtained with the constant message overhead assumption (Ulusoy, 1994). For high levels of replication, large number of messages need to be exchanged for update synchronization which, as we discussed before, leads to poor performance for carrier-sense protocols.

4. CONCLUSIONS

In this article, we have studied the effects of underlying network architecture on the performance of distributed RTDBSs. In particular, we have examined the relative performance of various network

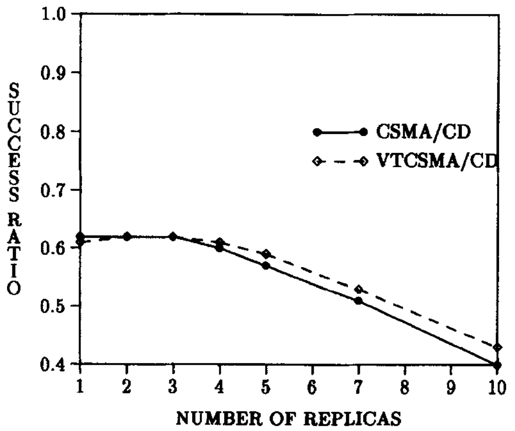

Figure 7. *Success_ratio* vs *N* (number of data replicas) for network access protocols CSMA/CD and VTCSMA/CD in an execution environment where update transactions predominate.

access protocols and analyzed the performance tradeoffs involved. We have also addressed the question of how realistic the assumption of constant network delay is for distributed RTDBSs.

A detailed simulation model of a distributed RTDBS used in an earlier work (Ulusoy, 1994) has been extended to capture the important features of a communication network. Real-time performance of distributed transactions has been evaluated in terms of the fraction of satisfied deadlines under two different network architecture types: carrier-sense multiple access networks and token ring networks. In addition to two conventional network access protocols (i.e., CSMA/CD and token ring), two real-time network access protocols (i.e., virtual time carrier-sense multiple access (VTCSMA/CD) and IEEE 802.5 token ring) have also been considered in our evaluations. The experiment results have shown that the real-time network access protocols, that involve timing constraints of communication messages in scheduling their channel access requests, do not necessarily yield better performance under all possible conditions. Performance of the protocols is highly dependent on current load and other characteristics of the distributed RTDBS. The real-time network access protocols help transactions meet their deadlines under high levels of transaction load. When the transaction load in the system increases, the difference between the performances obtained with real-time protocols and their nonreal-time counterparts becomes much more pronounced. The performance improvement provided by IEEE 802.5 over traditional token ring protocol has been observed to be at a higher level compared to the improvement of VTCSMA/CD over CSMA/CD. If the underlying execution environment is update-dominant, real-time protocols yield better performance when multiple copies of data items are being stored in the system. On the other hand, for query-dominant execution environments, the performance of the protocols is better than their nonreal-time counterparts only if at most a few copies of each data item is being stored. For all other conditions, which typically correspond to low loads of communication messages, it is not worthwhile to use a real-time network access protocol. Under such conditions, the performance benefit gained by exploiting real-time priorities is outweighed by the implementation overhead of those protocols.

Another interesting observation made in our experiments is that neglecting to model the underlying network in detail can lead to different conclusions. For various conditions tested, the carrier-sense and token ring network architectures have led to different performance results than those obtained with the constant message overhead assumption.

REFERENCES

Bernstein, P. A., and Goodman, N., An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases, *ACM Transactions on Database Systems* 9, 596-615 (1984).

Bernstein, P. A., Hadzilacos, V., and Goodman, N., *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.

Biyabani, S. R., Stankovic, J. A., Ramamritham, K., The Integration of Deadline and Criticalness in Hard Real-Time Scheduling, *9th Real-Time Systems Symposium*, 1988, pp. 152–160.

Bux, W., Local-Area Subnetworks: A Performance Comparison, *IEEE Transactions on Communications* 29, 1465–1473 (1981).

Garcia-Molina, H., Abbott, R. K., Reliable Distributed Database Management, *Proceedings of the IEEE 75*, 601–620 (1987).

Haritsa, J. R., Carey, M. J., Livny, M., Dynamic Real-Time Optimistic Concurrency Control, *11th Real-Time Systems Symposium*, 1990, pp. 94–103.

Haritsa, J. R., Carey, M. J., and Livny, M., Value-Based Scheduling in Real-Time Database Systems, Technical Report No. 1024, Dept. of Computer Science, University of Wisconsin-Madison, 1991.

Huang, J., Stankovic, J. A., Towsley, D., Ramamritham, K., Experimental Evaluation of Real-Time Transaction Processing, *10th Real-Time Systems Symposium*, 1989, pp. 144–153.

Lin, K. J., and Lin, M. ,2J., Enhancing Availability in Distributed Real-Time Databases, *ACM SIGMOD Record* 17, 34–43 (1988).

Özsu, M. T., and Niu, Y., Effects of Network Protocols on Distributed Concurrency Control Algorithm Performance, *4th International Conference on Computing and Information*, 1992, pp. 274–279.

Schwetman, H., CSIM: A C-Based, Process-Oriented Simulation Language, *Winter Simulation Conference*, 1986, pp. 387–396.

Sha, L., Rajkumar, R., Son, S. H., and Chang, C. H., A Real-Time Locking Protocol, *IEEE Transactions on Computers* 40, 793–800 (1991).

Sheth, A. P., Singhal, A., Liu, M. T., An Analysis of the Effect of Network Parameters on the Performance of Distributed Database Systems, *IEEE Transactions on Software Engineering* 11, 1174–1184 (1985).

Son, S. H., Chang, C. H., Performance Evaluation of Real-Time Locking Protocols Using a Distributed Software Prototyping Environment, *10th International Conference on Distributed Computing Systems*, 1990, pp. 124–131.

Soparkar, N., Levy, E., Korth, H. F., and Silberschatz, A., Adaptive Commitment for Real-Time Distributed Transactions, Technical Report TR-92-15, Department of Computer Science, University of Texas at Austin, 1992.

Token Ring Access Method, IEEE 802.5 Local Area Network Standard, IEEE Computer Society, Silver Spring, Maryland, 1985.

Ulusoy, Ö., and Belford, G. G., Real-Time Lock Based Concurrency Control in a Distributed Database System, *12th International Conference on Distributed Computing Systems*, 1992, pp. 136–143.

Ulusoy, Ö., Concurrency Control in Real-Time Database Systems, Technical Report UIUCDCS-R-92-1762, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

Ulusoy, Ö., Processing Real-Time Transactions in a Replicated Database Systems, *Journal of Distributed and Parallel Databases* 2, 405–436 (1994).

Zhao, W., Ramamritham, K., Virtual Time CSMA Protocols for Hard Real-Time Communication, *IEEE Transactions on Software Engineering* 13, 938–952 (1987).