ELSEVIER

A Study of Two Transaction-Processing Architectures for Distributed Real-Time Data Base Systems

Özgür Ulusoy

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

A real-time data base system (RTDBS) is designed to provide timely response to the transactions of data-intensive applications. Processing a transaction in a distributed RTDBS environment presents the design choice of how to provide access to remote data referenced by the transaction. Satisfaction of the timing constraints of transactions should be the primary factor to be considered in scheduling accesses to remote data. In this article, we describe and analyze two different alternative approaches to this fundamental design decision. With the first alternative, transaction operations are executed at the sites where required data pages reside. The other alternative is based on transmitting data pages wherever they are needed. Although the latter approach is characterized by large message volumes carrying data pages, it is shown in our experiments to perform better than the other approach under most of the work loads and system configurations tested. The performance metric used in the evaluations is the fraction of transactions that satisfy their timing constraints.

1. INTRODUCTION

Transactions processed in real-time data base systems (RTDBS) are associated with timing constraints, typically in the form of deadlines. Computer-integrated manufacturing, the stock market, banking, and command and control systems are several examples of RTDBS applications in which the timeliness of transaction response is as important as the consistency of data. In processing RTDBS transactions, it is very difficult to provide schedules guaranteeing all transaction deadlines. This difficulty comes from the consistency requirement of the underlying data base. The performance goal in RTDBS scheduling is to minimize the number of transactions that miss their deadlines.

Processing a transaction in a distributed RTDBS environment presents the design choice of how to provide access to remote data referenced by the transaction. In this article, we analyze two different alternatives to this fundamental design decision. The first alternative is the *distributed transaction* architecture, in which transaction operations are executed at the sites where required data pages[1] reside. The other alternative is the *mobile data* architecture, so named because, in this case, remote data pages required by a transaction are moved to the site of the transaction. A potential disadvantage of this approach is the communication overhead due to transmission of data pages between sites. However, the availability of new communication techniques that provide high-speed, large-volume data transfer reduces the communication overhead (Frieder, 1989). In both architectures, the primary factor considered in scheduling data accesses is the timing constraints of transactions.

This article presents a comprehensive simulation study that compares the performance of distributed RTDBS under those two different transaction-processing architectures. A detailed performance

Address correspondence to Prof. Özgür Ulusoy, Computer Engineering and Information Sciences, Bilkent University, Bilkent, Ankara 06533, Turkey.

[1] In both design approaches, a page is considered as the unit of buffering and data access.

0164-1212/95/$9.50
SSDI 0164-1212(94)00090-A

model of a distributed RTDBS was used in the evaluation of the architectures. The performance model captures the basic characteristics of a distributed data base system that processes transactions, each associated with a timing constraint in the form of a deadline. A unique priority is assigned to each transaction based on its deadline. The transaction-scheduling decisions are basically affected by transaction priorities. Various simulation experiments were carried out to study the relative performance of transaction-processing architectures under different work loads and system configurations. We also tried to find out how the locality of data references affects the performance of each architecture. The performance metric used in the evaluations is *success_ratio*, which gives the fraction of transactions that satisfy their deadlines.

To the best of our knowledge, no detailed investigation of transaction-processing architectures in RTDBS has been performed so far. As described in the following paragraphs, there have been some performance studies related to transaction scheduling in RTDBS; however, these studies were not specifically concerned with the performance of underlying transaction-processing architectures.

The first attempt to evaluate the performance of transaction-scheduling algorithms in RTDBS was provided in Abbott and Garcia-Molina (1988, 1989). The authors described and evaluated through simulation a group of real-time scheduling policies based on enforcing data consistency by use of a two-phase locking concurrency control mechanism. Huang et al. (1991) developed a new lock-based concurrency control protocol by combining some existing schemes to capitalize on the advantages of each of those schemes. Haritsa et al. (1990, 1992) studied, by simulation, the relative performance of two-well known classes of concurrency control algorithms (locking protocols and optimistic techniques) in an RTDBS environment. Agrawal et al. (1992) proposed a new locking approach, referred to as ordered sharing, which attempts to eliminate blocking of read and write operations in RTDBS. Son et al. (1992) examined a priority-driven locking protocol that decomposes the problem of concurrency control into two subproblems, namely, read-write synchronization and write-write synchronization, and integrates the solutions with two subproblems considering transaction priorities. In Kim and Srivastava (1991), new multiversion concurrency control algorithms were proposed to increase concurrency in RTDBS. We described several real-time concurrency control protocols and reported their relative performance in a single-site RTDBS (Ulusoy and Belford, 1993).

The remainder of the article is organized as follows. The next section describes the transaction-processing architectures studied. Section 3 provides the structure and characteristics of a distributed RTDBS model used in the evaluation of the architectures. Section 4 describes a set of experiments and our initial findings. Finally, Section 5 summarizes the conclusions of our work.

2. TWO ALTERNATIVE TRANSACTION-PROCESSING ARCHITECTURES

Two different architectures for processing RTDBS transactions are studied: distributed transaction (DT) and mobile data (MD). In the DT architecture, a transaction executes a cohort at each site that stores one or more data pages required by the transaction. This architecture was already studied for traditional distributed data base management systems by a number of researchers [e.g., Kohler and Jeng (1986), Garcia-Molina and Abbott (1987), Carey and Livny (1988)]. A distributed transaction was modeled as a collection of cohort processes to be executed at various data sites. As detailed in the next subsection, we extend this transaction model to a real-time environment in which the timing constraints of transactions are involved in scheduling local and remote data access requests of transactions.

The MD architecture, on the other hand, is based on transmitting data pages to wherever they are needed. This method is typically used in client/server data base management systems. In a client/server system, the data base resides on the server site, and items in the data base are accessed by application programs running on client sites (Wang and Rowe, 1991; Franklin et al., 1992). Data items required by the programs are shipped to the clients running the programs. We generalize this model to a distributed data base system in which each site can have its own data base and data items can be transferred among sites are needed. Timing constraints of transactions again play the major role in data access-scheduling decisions.

Both transaction-processing architectures described in the following subsections assume that there exists exactly one copy of each data page in the system.

2.1 DT Architecture

Each DT in this architecture exists in the form of a master process that executes at the originating site of the transaction and a collection of cohort processes that execute at various sites where the required data pages reside. Each transaction is assigned a globally unique priority based on its real-time constraint. This priority is carried by all of the cohorts of the transaction to be used in scheduling cohorts' executions. There can be at most one cohort of a transaction at each data site. If there exists any local data in the access list of the transaction, then one cohort is executed locally. 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 to find out which data site stores the data page referenced by the operation. A cohort process is initiated at that site (if it does not exist already) by the master process by sending an *initiate cohort* message to that site. If a cohort of the transaction already exists at that site, then it is activated only to perform the operation. Before accessing a data page, the cohort needs to obtain a lock on the page. In the case of a lock conflict (i.e., the lock has already been obtained by another cohort), if the lock-holding cohort has higher priority than the priority of the cohort that is requesting the lock, then the latter cohort is blocked. Otherwise, the lock-holding cohort is aborted and the lock is granted to the high-priority lock-requesting cohort. There is no possibility of blocking deadlock, because a high-priority transaction is never blocked by a lower priority transaction. After the successful completion of an operation, the result of the operation is sent to the master process, and the next operation of the transaction is executed by the appropriate cohort. When the last operation is completed, the transaction can be committed.

Upon the abort of a cohort, a message is sent to the master process of the aborted cohort to restart the whole transaction. The master process notifies the schedulers at all relevant sites to cause the cohorts of that transaction to abort. Then it waits for abort confirmation messages from each of these sites. When all the abort messages are received, the master can restart 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. We used the centralized two-phase commit protocol (Bernstein et al., 1987) for the atomic commitment of the distributed transactions. For the commitment of a transaction T, the master process of T is designated as *coordinator*, and each cohort process executing T's operations acts as a *participant* at its site. Following the execution of the last operation of transaction T, the coordinator (i.e., the master process of T) initiates phase 1 of the commit protocol by sending a *vote-request* message to all participants (i.e., cohorts of T) and waiting for a reply from each of them. If a participant is ready to commit, then it votes for commitment; otherwise, it votes for abort. An abort decision terminates the commit protocol for the participant. After collecting the votes of all participants, the coordinator initiates phase 2 of the commit protocol. If all participants vote for commit, then the coordinator broadcasts a *commit* message to them; otherwise, if any participant's decision is abort, then it broadcasts an *abort* message to the participants that voted for commit. If a participant, waiting for a message from the coordinator, receives a *commit* message, then the execution of the cohort of T at that site finishes successfully. After the successful commit of T, each cohort can write its updates (if any) into the local data base of its site. An *abort* message from the coordinator causes the cohort to be aborted. In that case, the data updates performed by the cohort are simply ignored.

The blocking delay of two-phase commit (i.e., the delay experienced at both the coordinator site and each of the participant sites while waiting to receive messages from each other) is explicitly simulated in conducting the performance experiments.

2.2 MD Architecture

This architecture is characterized by the movement of data pages among the sites. With this approach, each transaction is executed at a single site (the site at which it originated). Whenever a remote data page is needed by a transaction, the page is transferred to the site of the transaction. Besides the global data dictionary, which shows the origin of each data page in the system, each data site also maintains a relocation table to keep track of the pages transferred from/to that site. More specifically, for each data page P whose origin is site S_i and current location is site S_j, a record is maintained in the relocation table of each of the sites S_i and S_j. The record in the relocation table of S_i shows that P has been sent to S_j, and the record in the relocation table of S_j shows that P has been transferred from S_i.

Similar to the DT architecture, the operations of a transaction are executed one at a time. For each operation of a transaction T executed at site S_i, the data dictionary of S_i is referred to to find out the origin of the required data page P. If page P originated at site S_i but currently resides at another site, then a request message is sent to that site. If P has a remote origin, say, site S_j, and its current location is not S_i, then a request message is sent to S_j. The message includes the id of transaction T, its priority, the id of originating site S_i, and the id of the requested data page P. If P has been shipped to another site S_k, then the request message is forwarded to S_k.

Similar to DT, access to a data page is controlled on the basis of transaction priorities. Transaction T can obtain a lock on a page only if either the page is not being accessed by any other transaction, or T's priority is higher than the priority of the transaction currently accessing the page.[2] If the lock is granted, then the reply message contains both the grant and the requested page; otherwise, the message causes the transaction to become blocked until the requested lock becomes available. When the execution of a transaction finishes successfully, it can be committed locally. All updates performed by the transaction are stored on the local disk.

2.2.1 Management of relocation tables. Whenever a data page P with originating site S_i is transmitted to site S_j, the relocation tables at both sites are updated to keep track of the relocation information. A record is inserted into the relocation table of S_i to store the current location of P (i.e., S_j). The corresponding record inserted into the relocation table of S_j stores the origin of P (i.e., S_i). If page P later needs to be transmitted to another site S_k, then the related record is removed from the relocation table of S_j, and the id of originating site S_i is sent to S_k within the message containing data page P. Upon receiving that message, a new record is inserted into the relocation table of S_k. Another message from site S_j is sent to site S_i containing the new location of P so that the related record of the relocation table of S_i can be updated appropriately. It is ensured that the current location of a data page can always be found out by communicating with the originating site of that page.

3. DISTRIBUTED RTDBS MODEL

This section provides the model of a distributed RTDBS that we used to evaluate the transaction-processing architectures described in the preceding section. In the distributed system model, a number of data sites are interconnected by a local communication network. Each site contains a transaction generator, a transaction manager, a resource manager, a message server, a scheduler, and a buffer manager.

The transaction generator is responsible for generating the work load for each data site. The arrivals at a data site are assumed to be independent of the arrivals at the other sites. Each transaction in the system is distinguished by a globally unique transaction id. The id of a transaction is made up of two parts: a transaction number, which is unique at the originating site of the transaction, and the id of the originating site, which is unique in the system.

Each transaction is characterized by a real-time constraint in the form of a deadline. The transaction deadlines are *soft*; i.e., each transaction is executed to completion even if it misses its deadline. The transaction manager at the originating site of a transaction assigns a real-time priority to the transaction based on the earliest-deadline-first priority assignment policy; i.e., a transaction with an earlier deadline has higher priority than a transaction with a later deadline. If any two transactions originating from the same site carry the same deadline, then a scheduling decision between those two transactions prefers the one that has arrived earlier. To guarantee the global uniqueness of the priorities, the id of the originating site is appended to the priority of each transaction. The transaction manager is responsible for the implementation of any of the transaction-processing architectures (i.e., DT or MD) described in the preceding section. With the MD architecture, the management of the relocation table at each site is also the responsibility of the transaction manager.

There is no globally shared memory in the system, and all sites communicate via message exchanges over the communication network. A message server at each site is responsible for sending/receiving messages to/from other sites.

With the DT architecture, 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 pages as before. The MD architecture, on the other hand, does not need to use an atomic commitment protocol, because each transaction is executed locally.

IO and CPU services at each site are provided by the resource manager. IO service is required for reading or updating data pages, whereas CPU ser-

[2] This leads to a priority abort; the low-priority transaction currently accessing the page is aborted.

vice is necessary for processing data pages, performing various page access control operations (e.g., conflict check, locking, etc.), and processing 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 CPUs at each site. The CPU can be released by a transaction (or a cohort in the DT architecture) either resulting from a preemption, or when the transaction commits, or it is blocked/aborted because of a data conflict, or when it needs an IO or communication service. Communication messages are given higher priority at the CPU than other processing requests.

Reliability and recovery issues are not addressed here. We assumed a reliable system, in which no site failures or communication network failures occur. Also, we did not simulate in detail the operation of the underlying communication network. It was simply considered as a switching element between sites with a certain service rate.

Data transfer between disk and main memory is provided by the buffer manager. The FIFO page replacement strategy is used in the management of memory buffers.

3.1 Distributed RTDBS Model Parameters

The set of parameters described in Table 1 is used in specifying the configuration and work load of the distributed RTDBS. It is assumed that each site has one CPU and one disk. The seek time at each disk access is chosen randomly between 0.5 * *DiskSeekTime* and 1.5 * *DiskSeekTime*. Parameters *LocalitySetSize* and *LocalityProb* are used to study the impact of locality of data pages on the performance of the system. Section 4.3 is devoted to evaluating the effects of locality. The mean interarrival time of transactions to each of the sites is determined by the parameter *IAT*. Arrivals are assumed to be Poisson. The number of pages to be accessed by a transaction is determined by use of the parameter *XactSize*. The distribution of the number of pages is exponential. *SlackRate* is the parameter used in assigning deadlines to new transactions (see the next section).

3.2 Deadline Calculation

The slack time of an RTDBS transaction specifies the maximum length of time the transaction can be delayed and still satisfy its deadline. In our system, the transaction generator chooses the slack time of a transaction randomly from an exponential distribution with a mean of *SlackRate* times the estimated minimum processing time of the transaction. Although 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

Table 1. Distributed RTDBS Model Parameters

Parameter	Definition
Configuration	
NrOfSites	Number of sites in the system
DBSize	Data base size at each site (pages)
MemSize	Size of the memory buffers used to hold data pages at each site (pages)
PageSize	Page size (bytes)
CPURate	Instruction rate of CPU at each site (million instructions per second)
InstrProcessPage	Number of instructions to process each page
DiskSeekTime	Average disk seek time (milliseconds)
DiskTransTime	Disk transfer time of one page (milliseconds)
InstrInitDisk	CPU cost of initializing a disk access (instructions)
NWBandwidth	Network bandwidth (mega bits per second)
CtrlMesSize	Size of a control message (bytes)
InstrInitMes	CPU cost to initialize sending/receiving a message (instructions)
InstrPerMesByte	CPU cost of sending/receiving each byte of a message (instructions)
LocalitySetSize	Size of the set of the most recently accessed pages at a site
LocalityProb	Probability of accessing a page in the locality set
Transaction	
IAT	Mean interarrival time of transactions at each site
XactSize	Average number of pages accessed by a transaction
UpdateRate	Probability of updating the accessed page
RemoteAccessRate	Probability of accessing a page with a remote origin
InstrStartXact	Number of instructions to initialize a transaction
InstrEndXact	Number of instructions to terminate a transaction
SlackRate	Average rate of slack time of a transaction to its processing time

deadline of a transaction T is determined by the following formula:

$$deadline(T) = start\_time(T) + minimum\_processing\_time\_estimate(T) + slack\_time(T)$$

where

$$slack\_time(T) = expon(SlackRate * minimum\_processing\_time\_estimate(T))$$

The estimated minimum processing time formula determines the processing time of a transaction under an ideal execution environment in which the system is unloaded (i.e., no data and resource conflicts occur among transactions), and the transaction does not require any data page that is remotely placed. To satisfy the deadline, the delay experienced by the transaction due to conflicts and remote accesses should not exceed the slack time included in the deadline formula.

$$minimum\_processing\_time\_estimate(T) = CPU\_delay(T) + IO\_delay(T)$$

Let *Pages*(T) denote the actual number of pages accessed by transaction T,

$$CPU\_delay(T) = \frac{10^{-3}}{CPURate} * (InstrStartXact + (1 + UpdateRate) * Pages(T) * InstrProcessPage + InstrEndXact)$$

$$IO\_delay(T) = \left[\left(1 - \frac{MemSize}{DBSize}\right) * Pages(T) * \left(\frac{InstrInitDisk}{CPURate} * 10^{-3} + DiskSeekTime + DiskTransTime\right)\right] + \left[UpdateRate * Pages(T) * \left(\frac{InstrInitDisk}{CPURate} * 10^{-3} + DiskSeekTime + DiskTransTime\right)\right]$$

The expression contained in the second pair of square brackets corresponds to the delay experienced while writing updated pages back into the disk. The unit of both *CPU_delay*(T) and *IO_delay*(T) is milliseconds.

Table 2. Distributed RTDBS Model Parameter Values

Parameter	Value
NrOfSites	10
DBSize	1250 pages
MemSize	200 pages
PageSize	4 Kbytes
CPURate	30 million instructions per second
InstrProcessPage	30,000 instructions
DiskSeekTime	20 milliseconds
DiskTransTime	2 milliseconds
InstrInitDisk	5,000 instructions
NWBandwidth	10 mega bits per second (e.g., Ethernet), 100 mega bits per second (e.g., Fiber Distributed Data Interface)
CtrlMesSize	256 bytes
InstrInitMes	20,000 instructions
InstrPerMesByte	3 instructions
IAT	400 milliseconds
XactSize	10 pages
UpdateRate	0.5
RemoteAccessRate	0.5
InstrStartXact	30,000 instructions
InstrEndXact	40,000 instructions
SlackRate	10

4. PERFORMANCE EVALUATION

The details of the distributed RTDBS model and the transaction-processing architectures described in previous sections were captured in a simulation program. The values of configuration and work load parameters common to all simulation experiments are presented in Table 2. All data sites in the system are assumed identical and operate under the same parameter values. The settings used for resource-related parameters were basically taken from the experiments of Franklin et al. (1992).[3] Those values can be accepted as reasonable approximations of what can be expected from today's systems. The work load parameters were selected to provide a transaction load and data contention high enough to bring out the differences between the performances of transaction-processing architectures. The high transaction load was obtained by setting the average interarrival time parameter (i.e, *IAT*) to a relatively small value that leads to CPU and IO utilizations of $> 90\%$. High levels of data contention were obtained by considering a relatively small data base size at each site (i.e., *DBSize*). This small data base can be considered as the most frequently accessed fraction of a larger data base. Under low transaction loads or when data conflicts among transactions were few, both architectures were observed to be

[3]There are a few differences between their values and ours, because their simulator was designed for a client/server DBMS architecture.

equally successful in satisfying the timing constraints of almost all transactions.

The performance metric used in the evaluation of the architectures is *success_ratio*, i.e., the fraction of transactions that satisfy their deadlines. The other important performance metrics that helped us analyze the results are the average number and volume of messages required to execute a transaction and the average network delay and IO delay experienced by each transaction. In simulating the MD architecture, it is assumed that each data message contains only one data page.

The simulation program was written in CSIM (Schwetman, 1980), which is a process-oriented simulation language based on the C programming language. For each configuration of each experiment, the final results were evaluated as averages over 25 independent runs. Each configuration was executed for 500 transactions originating at each 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.

4.1 Varying Remote Data Access Rate

In this experiment, we investigated various performance characteristics of transaction-processing architectures under different levels of remote data accesses issued by transactions. The level of remote data accesses is determined by the parameter *RemoteAccessRate* and corresponds to the fraction of data pages of remote origin in the set of all data pages accessed by a transaction. It is assumed that remote data accesses are uniformly distributed among all remote sites (i.e., site of the remote data is chosen randomly).

The first set of results examined in this section is that of the resource requirements experienced by each transaction under architectures DT and MD. Those results help us analyze the relative performance of the studied architectures. Figures 1 and 2 present, respectively, the average values of the number and the total volume (in bytes) of messages exchanged between sites for each transaction. With architecture DT, more messages are involved in controlling the execution of a transaction. As detailed in Section 2.1, the master process of a transaction needs to send an *initiate cohort* message to each site where a cohort of the transaction is executed. The execution of a transaction operation at a remote site is started on receiving an *activate* message from the master process of the transaction, and the result of the operation is sent back to the master

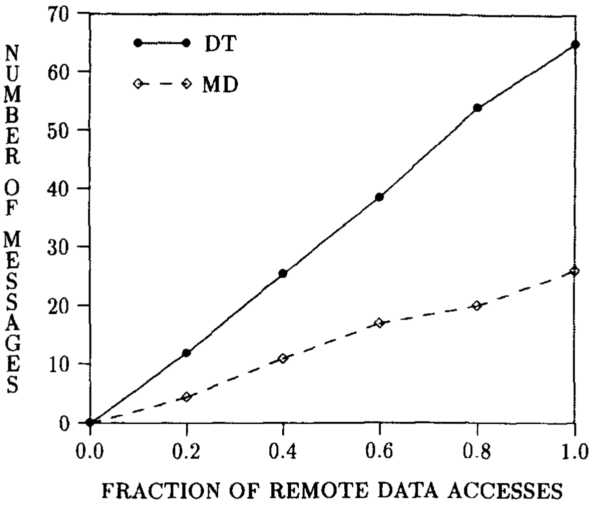

Figure 1. Average number of messages sent per transaction as a function of the level of remote data accesses.

process within an *operation complete* message. The atomic commitment of a transaction also requires a couple of messages to be exchanged between the master process and each of the remote cohorts of the transaction. With architecture MD, a request message is generated for each operation accessing a remote page,[4] and the reply message contains the requested page. There is no need to execute an atomic commitment protocol with MD; transactions can be committed locally without requiring communication with other sites.

Another factor that has a considerable influence on the relative number of messages generated with both architectures is the priority abort of transactions resulting from priority-based page access control. With DT, when a cohort of a transaction is aborted, the master process of the transaction should send control messages to the sites executing the cohorts of the transaction to notify them about the abort decision. Also, when the aborted transaction is restarted, the master process again requires to communicate with other sites to perform remote accesses, although it might already have communicated with them before being aborted. With MD, on the other hand, a restarted transaction can find the previously accessed data pages in local buffers; thus, it is not required to generate new request messages.

Although more messages need to be exchanged with DT for the execution of each transaction, the total volume of those messages is less than the message volume of a transaction with the MD ap-

[4] If the requested page is not residing at its originating site, then the message is forwarded to the current site of the page.

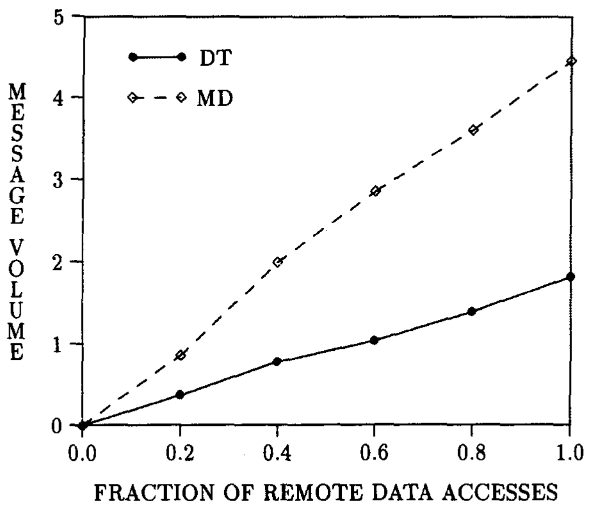

Figure 2. Average message volume (Kbytes) per transaction as a function of the level of remote data accesses.

proach (Fig. 2). All messages associated with DT are control messages (256 bytes), whereas with MD, both control messages and data messages (containing four-Kbyte pages) are exchanged between sites.

The overhead of messages (in terms of both network delay and CPU time used for processing messages) per transaction was also measured with the DT and MD architectures. It was observed that if a slow network is used, then the overall message cost of a transaction does not show much difference under different architectures. Figure 3 displays the average values of the network delay, the CPU delay, and the overall (network + CPU) delay of messages issued for a transaction with both DT and MD. The network delay values were obtained for a slow network (i.e., with *NWBandwidth* = 10 mega bits per second). The primary CPU cost of a message is the initialization time experienced at the source (destination) site for transmitting (receiving) the message. Because more messages are generated with the DT architecture, the CPU cost of the messages is higher. On the other hand, higher volume of messages with the MD architecture results in greater network delay for each transaction. The overall overhead of messages with MD was shown to be comparable to that of DT; however, when the experiment was repeated with a faster network (i.e., by setting *NWBandwidth* to 100 mega bits per second), MD was observed to provide lower message delay (Figure 4). With a fast network, the CPU cost of messages plays the major role in determining the average delay of messages for a transaction.

Another resource requirement of transactions is the disk access to read/write data pages. The impact of the overhead of disk accesses on the relative performance of transaction-processing architectures was also investigated. Examining Figure 5, one can see that use of MD considerably reduces the disk access delay of a transaction experienced with DT. The values presented in the figure include both the delay of transferring data from/to disk and waiting times at the disk queues. If all accesses are local, then there is no difference between disk access delays of DT and MD. As the friction of remote data accesses increases, MD produces lower disk access times for transactions. Remember that all the updates of a transaction are written to the local disk together at the commit time of the transaction. With DT, each remote data page updated by the transac-

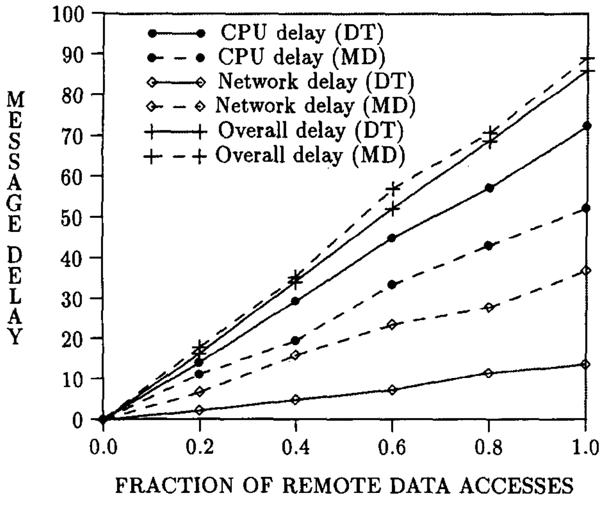

Figure 3. Average delay (milliseconds) of messages for a transaction with the slow network (*NWBandwidth* = 10 mega bits per second).

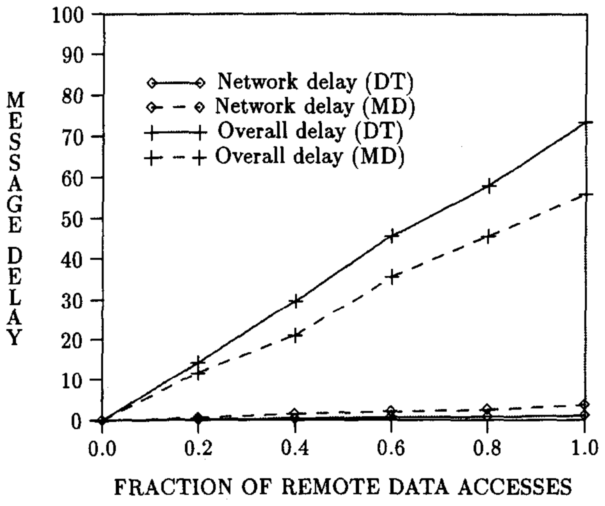

Figure 4. Average delay (milliseconds) of messages for a transaction with the fast network (*NWBandwidth* = 100 mega bits per second).

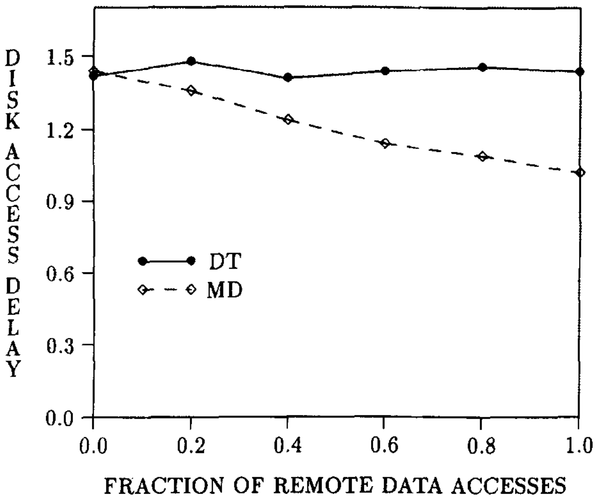

Figure 5. Average disk access delay (seconds) for a transaction.

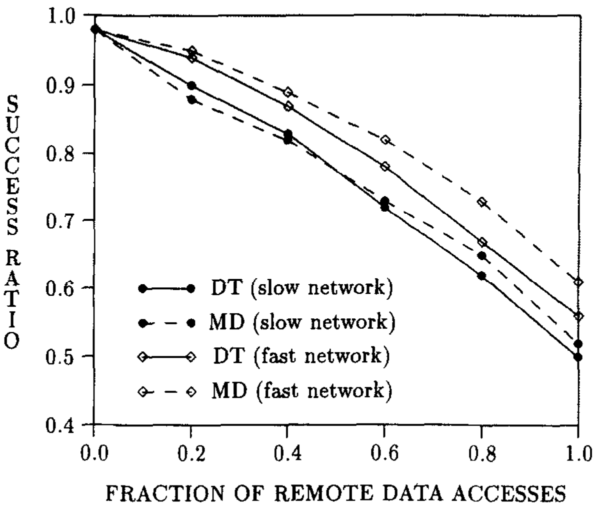

Figure 6. Real-time performance in terms of *success_ratio* (the fraction of transactions that satisfy their deadlines) under both a slow and a fast network.

tion is restored to the disk of data page's site. A separate disk access is required at each site storing the pages updated by the transaction. With MD, on the other hand, the updated remote pages can be consecutively placed on the local disk preventing the delay of separate seek time for each stored page. The seek time constitutes the major delay of a disk access (the value used in our experiments is *DiskSeekTime* = 20 milliseconds).

With the resource requirement results in mind, we now turn to the resulting real-time performance of the transaction-processing architectures. The *success_ratio* results with both a slow network (*NWBandwidth* = 10 mega bits per second) and a fast network (*NWBandwidth* = 100 mega bits per second) are presented in Figure 6. When all the pages accessed by each transaction are local, there is no difference between the performances of the architectures. Because the remote accesses are handled in different ways by the architectures, the difference between their performances appears when remote accesses are also considered for the transactions. As more remote pages are accessed more transactions miss their deadlines with both architectures because of the involvement of communication messages. If the underlying network is slow, then the real-time performances of DT and MD are comparable to each other. Under high levels of remote data accesses, MD provides a slight improvement over DT. Although each transaction is characterized by lower resource requirements (in terms of disk access delay and the number of messages exchanged among sites to control transaction execution) with MD, the higher volume of messages due to the transmission of data pages prevents MD from being the clear winner. However, if the slow network is replaced by a faster one, the message delay will no longer be a bottleneck. As displayed in Figure 6, with a fast network, DT cannot reach the real-time performance level attained by MD. The difference between the number of satisfied deadlines provided with each architecture increases as the fraction of remote accesses increases. This observation directly follows the message and IO delay results obtained with a fast network. The relative real-time performance of the architectures is primarily determined by the resource requirements of processed transactions.

4.2 Evaluating Architectures Under a Nonreal-Time Environment

It was shown in the previous section that architecture MD is preferable to DT in processing transactions with real-time constraints (i.e., deadlines). The performance of the architectures was evaluated in terms of the fraction of satisfied transaction deadlines. To see whether there might be any differences in the performance results if the transactions processed are not characterized by timing constraints, we repeated the experiments in an environment in which no real-time priority information is involved in scheduling data accesses of transactions. The two-phase locking scheme is used in controlling concurrent accesses to data pages. The performance metric used in the evaluations is the average response time of transactions.

The results obtained with architectures DT and MD are displayed in Figure 7. Again, two different

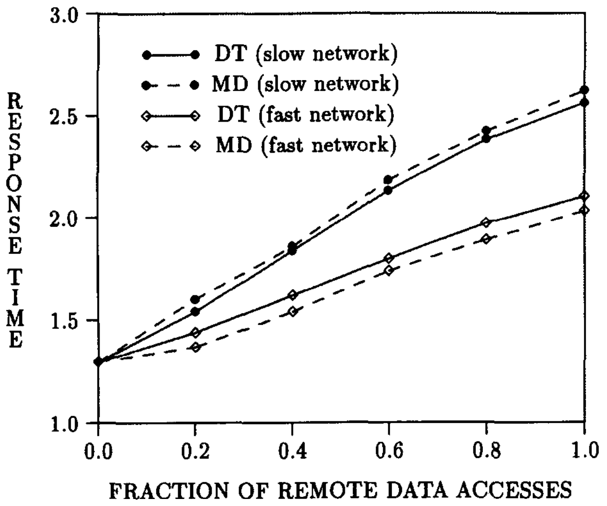

Figure 7. Average response time (seconds) of transactions in a nonreal-time environment.

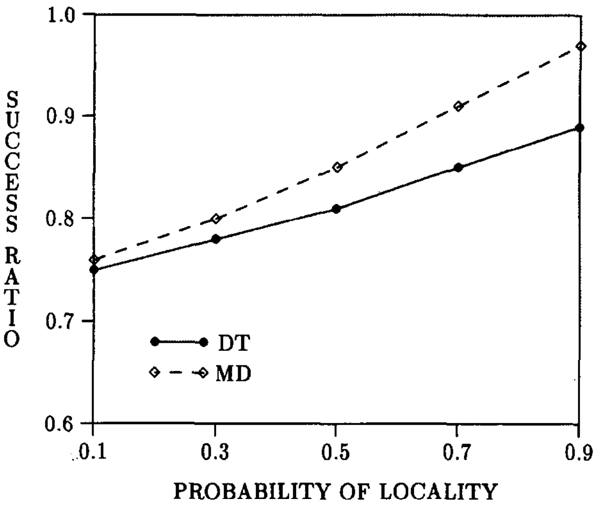

Figure 8. Real-time performance in terms of *success_ratio* (the fraction of transactions that satisfy their deadlines) as a function of the locality of page references.

networks with *NWBandwidth* = 10 and 100 mega bits per second, respectively, were used in the evaluations. With the slow network, DT performed a little bit better (i.e., produced lower average response time) than MD. With the fast network, on the other hand, MD achieved better performance; however, if we compare the results with those presented in the previous section, the performance improvement provided by MD over DT, in this case, is very limited. It can be concluded that MD is not superior to DT in a real-time environment. One reason is the fact that no priority aborts occur in a nonreal-time environment, which lead to much higher message overhead with DT than with MD, as explained before. Also, with MD, the updates of a transaction are written to disk together, therefore another transaction in the IO queue has to wait until all those writes are completed. On the other hand, in processing real-time constrained transactions, IO queues are organized on the basis of transaction priorities. Thus, a high-priority transaction can preempt a lower priority transaction writing its updates. The preemption can help the high-priority transaction terminate as soon as possible, whereas the low-priority transaction can still have enough time to satisfy its deadline. This might be another factor leading to the different results obtained in two different environments with separate performance metrics.

4.3 Sensitivity to the Page Access Locality

So far, the locality concept was not considered in the experiments, and data pages accessed by each transaction were chosen on a random basis. In the experiment discussed in this section, we tested the sensitivity of real-time performance results to the locality of page references. To model page reference locality, we used the locality set concept introduced in Wang and Rowe (1991). Parameters *LocalitySetSize* and *LocalityProb* are used to model locality. The locality set of a site is defined as the last x pages accessed by the most recent transactions originating at that site, and x is the value of the parameter *LocalitySetSize*. The parameter *LocalityProb* specifies the probability that a page accessed by an active transaction is in the locality set.

The results displayed in Figure 8 were obtained by setting *LocalitySetSize* to 30 pages. The experiment was performed assuming a slow network (*NWBandwidth* = 10 mega bits per second) and setting the probability of accessing a page with a remote origin (*RemoteAccessRate*) to 0.5. The value of *LocalityProb* varied from 0.1 (corresponding to a low locality) to 0.9 (very high locality) in increments of 0.2. Increasing the locality of page accesses results in better performance with both architectures. For high values of locality, because each page referenced by a transaction has most probably been accessed recently, it is likely that the page can be found in memory buffers. This prevents the disk access delay, which is a substantial overhead in transaction execution. As can be seen from Figure 8, MD benefits more from increasing locality. This result is due to the fact that, with MD, recently accessed pages with remote origin, as well as the local ones, can be found in local memory buffers. As a result, when such a page needs to be reaccessed, no communication with remote sites is required. With DT, on the other hand, each remote data page should be processed at its site; thus, the locality cannot prevent the overhead of messages exchanged to control the execution of remote operations.

The relative performance results obtained with some other settings of *LocalitySetSize* were very similar to those just discussed; thus, they are not displayed here.

4.4 Varying the Page Size

In this experiment, we studied the impact of the page size on the real-time performance of the system. The values of parameters *InstrProcessPage* (i.e., number of instructions to process a page) and *DiskTransTime* (i.e., disk transfer time of a page) were assumed to be proportional to the page size and determined on the basis of the current value of *PageSize*. The values of *XactSize* (i.e., average transaction size in pages) and *DBSize* (i.e., number of pages stored in the data base of each site) were kept constant while the performance was being measured with different page sizes.

The performance obtained with architectures DT and MD under various page sizes are presented in Figure 9. Similar to the previous experiment, the results were obtained by operating the system with a slow network (*NWBandwidth* = 10 mega bits per second) and with a remote data access probability (*RemoteAccessRate*) of 0.5. Because the average number of pages accessed by each transaction remains the same, the resource requirements of transactions (in terms of the CPU time, disk, and network accesses) increase as the size of a page increases. The higher resource contention among transactions results in a decrease in performance; i.e., fewer transactions can satisfy their deadlines as the accessed pages become larger. The page size has a greater impact on the performance with architecture MD. Large page sizes lead to more communication overhead for MD because data messages containing pages as well as short control messages need to be exchanged among sites in controlling transaction execution. MD performs well under small page sizes; however, DT seems to be preferable if the system has a large page size.

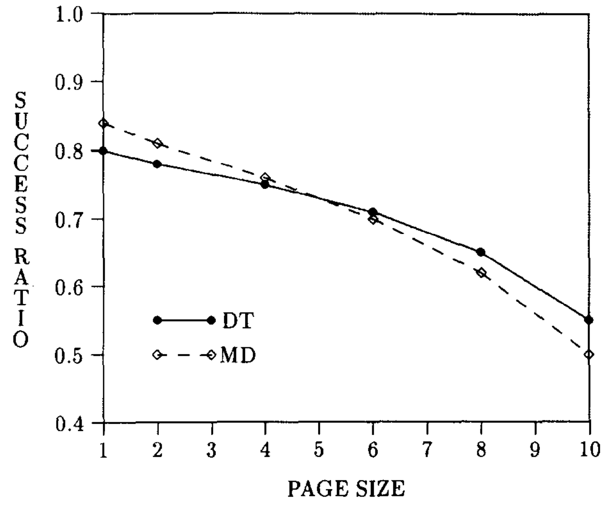

Figure 9. Real-time performance in terms of *success_ratio* (the fraction of transactions that satisfy their deadlines) as a function of *PageSize* (Kbytes).

5. CONCLUSIONS

In this article, we described two different transaction-processing architectures for distributed RTDBS and evaluated their performance under various work loads and system configurations. The primary performance consideration in an RTDBS (i.e., a data base system that processes transactions with timing constraints) is to provide schedules that maximize the number of satisfied timing constraints. We investigated how successful each transaction-processing architecture is in achieving that performance goal.

The first architecture analyzed, DT, distributes the execution of each transaction onto the sites that store the data pages required by the transaction. The other architecture, MD, moves the remote data pages requested by a transaction to the site of the transaction. The main drawback of DT is the large number of messages required to control the execution of a distributed transaction, whereas the primary overhead of MD is the large-sized messages carrying data pages between sites. Both architectures consider the timing constraints of transactions in scheduling accesses to data and hardware resources.

To analyze the effectiveness of the transaction-processing architectures in satisfying timing constraints, we built a performance model of a distributed RTDBS. Various experiments were conducted by use of a simulation program developed on the basis of the performance model. The main conclusions of the experiments are as follows:

- The relative performance of the architectures is primarily determined by the resource requirements of transactions processed under each of the architectures. The results obtained in resource requirement experiments (in terms of the average number and volume of messages required to execute a transaction and the average network delay and IO delay experienced by each transaction) helped explain the behavior of the architectures under various levels[5] of remote data accesses.

[5]The level of remote data accesses corresponds to the fraction of remote data pages accessed by a transaction.

With a slow network, the overhead of messages for each transaction did not show much difference under two different architectures. Although the average message volume with MD was much higher, DT was not able to outperform MD because the cost of transferring a message is primarily due to the CPU time to initiate sending/receiving the message, not the transmission time; DT was characterized by the larger number of messages (compared to MD) issues for each transaction. When a fast network was used, MD demonstrated superior performance, especially under high levels of remote data accesses. The average volume of messages did not have any influence on the performance.

- To see how the performance results are affected when transactions have no timing constraints, the experiments were repeated by processing non-real-time transactions and using the average response time of transactions as the performance metric. In this case, no considerable performance improvement was provided by MD. The primary reason for that result is the fact that no priority aborts (due to timing constraints) occur in a non-real-time environment, which was shown to lead to much more message overhead with DT than with MD.
- We also investigated the effects of the locality of data references on the performance of each architecture. Increasing the locality resulted in better performance with both architectures DT and MD. However, MD was shown to benefit more from high locality due to storing recently accessed remote pages in local memory buffers.
- Although large page sizes affected both architectures negatively, the page size appeared to have a greater impact on the performance for MD when the system was operated with a slow network.

In summary, our results suggest that MD architecture should be preferred in distributed RTDBS unless the underlying network is very slow or the system is characterized by very large data pages.

REFERENCES

Abbott, R., and Garcia-Molina, H., Scheduling real-time transactions: A performance evaluation, in *14th International Conference on Very Large Data Bases,* 1988, pp. 1-12.

Abbott, R., and Garcia-Molina, H., Scheduling real-time transactions with disk resident data, in *15th International Conference on Very Large Data Bases,* 1989, pp. 385-396.

Agrawal, D., El Abbadi, A., and Jeffers, R., Using delayed commitment in locking protocols for real-time databases, in *ACM SIGMOD Conference,* 1992, pp. 104-113.

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

Carey, M. J., and Livny, M., Distributed concurrency control performance: A study of algorithms, distribution, and replication, in *14th International Conference on Very Large Data Bases,* 1988, pp. 13-25.

Franklin, M. J., Carey, M. J., and Livny, M., Global Memory Management in Client-Server DBMS Architectures, Computer Science Technical Report no. 1094, University of Wisconsin—Madison, Madison, Wisconsin, 1992.

Frieder, O., Communication Issues in Data Engineering: Have Bandwidth—Will Move Data, *IEEE Data Eng. Bull.* (1989).

Garcia-Molina, H., and Abbott, R. K., Reliable Distributed Database Management, *Proc. IEEE* 75, 601-620 (1987).

Haritsa, J. R., Carey, M. J., and Livny, M., On being optimistic about real-time constraints, *ACM SIGACT-SIGMOD-SIGART,* 1990, pp. 331-343.

Haritsa, J. R., Carey, M. J., and Livny, M., Data Access Scheduling in Firm Real-Time Database Systems, *Real-Time Syst.* 4, 203-241 (1992).

Huang, J., Stankovic, J. A., Ramamritham, K., and Towsley, D., On using priority inheritance in real-time databases, in *12th Real-Time Systems Symposium,* 1991, pp. 210-221.

Kim, W., and Srivastava, J., Enhancing real-time DBMS performance with multiversion data and priority based disk scheduling, in *12th Real-Time Systems Symposium,* 1991, p. 222-231.

Kohler, W. H., and Jeng, B. H., Performance evaluation of integrated concurrency control and recovery algorithms using a distributed transaction testbed, in *6th International Conference on Distributed Computing Systems,* 1986, pp. 130-139.

Schwetman, H., CSIM: A C-based, process-oriented simulation language, in *Winter Simulation Conference,* 1986, pp. 387-396.

Son, S. H., Park, S., and Lin, Y., An integrated real-time locking protocol, in *8th International Conference on Data Engineering,* 1992, pp. 527-534.

Ulusoy, Ö., and Belford, G. G., Real-Time Transaction Scheduling in Database Systems, *Infor. Syst.* 18, 559-580 (1993).

Wang, Y., and Rowe, L. A., Cache consistency and concurrency control in a client/server DBMS architecture, in *ACM SIGMOD Conference,* 1991, pp. 367-376.