ELSEVIER

The Journal of Systems and Software 42 (1998) 247–262

The Journal of Systems and Software

Transaction processing in distributed active real-time database systems

Özgür Ulusoy [1]

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

Received 1 July 1997; received in revised form 1 November 1997; accepted 1 January 1998

Abstract

An active real-time database system (ARTDBS) is designed to provide timely response to the critical situations that are defined on database states. Although a number of studies have already addressed various issues in ARTDBSs, little attention has been paid to scheduling transactions in a distributed ARTDBS environment. In this paper, [2] we describe a detailed performance model of a distributed ARTDBS and investigate various performance issues in time-cognizant transaction processing in ARTDBSs. The experiments conducted evaluate the performance under various types of active workload and different distributed transaction-processing architectures. The performance metric used in the evaluations is the fraction of transactions that violate their timing constraints. We also describe and evaluate a nested transaction execution scheme that improves the real-time performance under high levels of active workload.

Keywords: Active real-time database systems; Transaction scheduling; Nested transactions; Performance evaluation

1. Introduction

Real-time database systems are designed to provide timely response to the transactions of data-intensive applications. Each transaction processed in a real-time database system is associated with a timing constraint typically in the form of a deadline. Active database systems, on the other hand, extend conventional database systems with the ability to specify and implement reactive behavior which is typically specified in terms of event-condition-action (ECA) rules. The general form of an ECA rule is: on *event* if *condition* do *action*. The semantics of such a rule is that when an event occurs, the corresponding condition is checked, and if the condition is satisfied, then a specified action is executed [16]. Therefore, an active database system has to monitor events of interest and detect their occurrences. The semantics of rule execution can be specified in a transaction framework. A transaction which triggers rules can be called a *triggering transaction*, and the transaction which executes the triggered rule can be called the *triggered transaction*.

Active real-time database systems (ARTDBSs) field has emerged as a result of the requirement to apply real-time scheduling techniques to rule execution in active database systems that support time-critical applications. Some examples of such application areas are command and control systems, automated manufacturing, air-traffic control, intelligent network services, and cooperative distributed navigation systems [42]. Few studies, that are briefly described in the next section, have addressed various problems that might arise in integrating the concepts from real-time scheduling algorithms and active database systems.

In our work, we investigate various performance issues in time-cognizant transaction processing in distributed ARTDBSs. We describe a performance model designed for studying the performance of various components of a distributed ARTDBS, and present the results of experiments conducted to be able to address the performance issues in executing transactions under various types of active workload. The performance metric used in the evaluations is the fraction of transactions that miss their deadlines. Two different transaction-processing architectures, that we call *distributed transaction* and *mobile data*, are employed in the experiments. In the distributed transaction architecture, a transaction executes a cohort at each site that stores one or more data pages required by the transaction. The mobile data

[1] E-mail: oulusoy@bilkent.edu.tr.

[2] An earlier version of this paper appears in the Proceedings of the Second International Workshop on Active, Real-Time and Temporal Database Systems.

0164-1212/98/$19.00
PII: S0164-1212(98)10013-4

architecture, on the other hand, is based on transmitting data pages to wherever they are needed. We also consider different types of semantics in controlling the concurrent execution of triggering and triggered transactions. Besides conducting experiments with the assumption that the triggering and triggered transactions share the locks, we also describe and evaluate a nested transaction execution scheme that exploits the nested structure of rule execution to improve the performance.

1.1. Related work

Most of the work in active database systems area to date has concentrated on rule specification, efficient event detection and rule execution (e.g., [6,8,10,17, 18,21–23,29]). A detailed description of the issues related to active database systems can be found in Ref. [37]. Involvement of timing constraints in active databases was first considered in the HiPAC (High Performance ACtive Database System) project [9,15]. Three basic concepts explored in this project are active database management, timing constraints, and contingency plans. *Contingency plans* are alternate actions that can be invoked whenever the system determines that it cannot complete an action within its deadline. A knowledge model was developed for the project that provides primitives for defining condition-action rules and timing constraints, control mechanisms for efficient rule searching, and support for the execution model. The execution model was provided to specify the semantics of rule execution in a transaction framework. An important issue determined by the execution model is the *coupling mode* between the triggering transaction and the triggered rule. Three basic coupling modes are introduced in Ref. [16]: *immediate*, *deferred*, and *detached*. In immediate coupling mode, the triggered rule is executed immediately within the triggering transaction. In deferred coupling mode, the triggered rule is executed at the end but before the commit of the triggering transaction. Finally, in detached coupling mode, the triggered rule is executed in a separate transaction independent of the triggering transaction.

The research on real-time database systems to date has mainly focused on development and evaluation of time-cognizant scheduling techniques that aim to maximize the fraction of satisfied timing constraints while maintaining consistency of the underlying database (e.g., [1,4,11,12,14,19,26–28,32,35,40,46]). For an overview of the recent research on real-time database systems, the reader is encouraged to refer to [36,49]. The results of a considerable amount of research devoted to various issues in *distributed* real-time database systems have also appeared in the literature. These issues include concurrency control (e.g., [32,43,45]), deadline assignment (e.g., [30,33]), replication (e.g., [47]), and commitment (e.g., [24,34,44]). In Ref. [45], several distributed real-time concurrency control protocols were described and the relative performance of the protocols was reported in a nonreplicated database environment. In Ref. [43], several methods were investigated to apply a real-time locking protocol, called *priority-ceiling*, as a basis for concurrency control in a distributed environment. The relative performance of static and dynamic locking approaches in a distributed real-time database system was studied in Ref. [32].

The problem of assigning priorities to transactions in ARTDBSs was studied by Purimetla et al. [38] and Sivasankaran et al. [42]. Three priority assignment policies, that use different amount of semantic information about transactions, were proposed, and the performance of the policies was evaluated using an ARTDBS simulator. The same research group also developed some strategies for data placement, logging, and recovery to achieve efficient transaction processing in ARTDBSs [41]. It was shown that exploiting the characteristics of data for transaction processing, placing the data at the appropriate level of the memory hierarchy, and performing logging and recovery of data appropriate for each type of data is crucial to attain high performance in ARTDBSs.

Branding and Buchmann [5] identified 'network management' as one of the applications that require both active and real-time database support. Their primary work was development of an ARTDBS for this application and seamless integration of the ARTDBS's execution model with the underlying operating system primitives.

Berndtsson and Hansson [2] characterized the basic features of active and real-time database systems, and addressed several issues that need to be considered while combining those features. In a recent work, Datta and Son [13] studied various concurrency control methods in ARTDBSs, and proposed a number of new strategies. Performance of the proposed concurrency control strategies was investigated through simulation experiments.

2. A distributed active real-time database system model

We have extended the performance model of a distributed real-time database system that we used in an earlier work [48], by adding a *Rule Manager* to handle triggering transactions. In the distributed system model, a number of data sites are interconnected by a local communication network. As depicted in Fig. 1, each site contains a *Transaction Generator*, a *Transaction Manager*, a *Rule Manager*, a *Message Server*, a *Resource Manager*, a *Scheduler*, and a *Buffer 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 in the system is distinguished by a globally unique transaction id. The id of a transaction is made up of two parts: a transaction

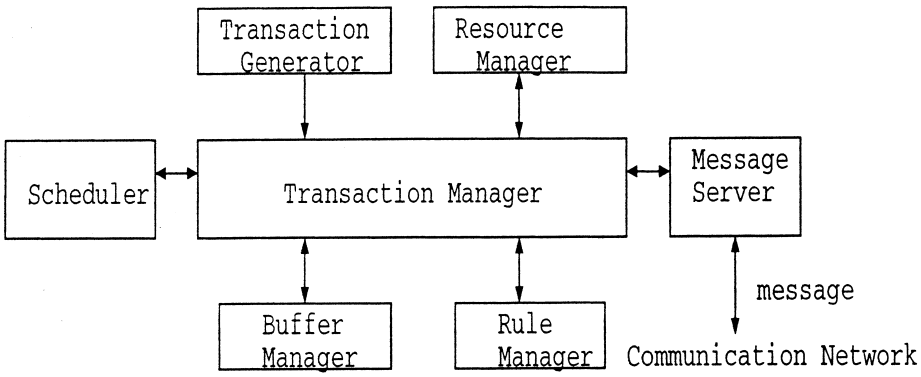

Fig. 1. Performance model used for each data site.

number which is unique at the originating site of the transaction and the id of the originating site which is unique in the system.

The Transaction Manager is responsible for modeling the execution of transactions. It accepts transactions from the Transaction Generator and the Rule Manager. 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 originated from the same site carry the same deadline, 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 described in Section 2.2. For each operation of the executing transaction, the Transaction Manager communicates with the Scheduler to see whether the operation leads to any conflict with the operations of the other transactions.

Access requests for data pages are ordered by the Scheduler on the basis of the concurrency control protocol executed. The protocol we use in our experiments is the *High-Priority* concurrency control protocol which resolves data conflicts always in favor of high-priority transactions [1]. At the time of a data lock conflict, if the lock-holding transaction has higher priority than the priority of the transaction that is requesting the lock, the latter transaction is blocked. Otherwise, the lock-holding transaction is aborted and the lock is granted to the high priority lock-requesting transaction. Assuming that no two transactions have the same priority, this protocol is deadlock-free since a high priority transaction is never blocked by a lower priority transaction. Concurrency control is implemented at a page granularity.

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. Reliability and recovery issues were not addressed in this paper. 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 just considered as a switching element between sites with a certain service rate.

I/O and CPU services at each site are provided by the Resource Manager. I/O service is required for reading or updating data pages, while CPU service 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 I/O queues are organized on the basis of real-time priorities, and preemptive-resume priority scheduling is used by the CPU's at each site. The CPU can be released by a transaction either due to a preemption, or when the transaction commits or it is blocked/aborted due to a data conflict, or when it needs an I/O or communication service.

Data transfer between disk and main memory is provided by the Buffer Manager. *The Least Recently Used* (*LRU*) page replacement strategy is used in the management of memory buffers.

2.1. Handling the active workload

The Transaction Manager informs the Rule Manager of each page update. Each successful update operation which changes the database state is considered as an *event*. The Rule Manager is responsible to check if any *action* is triggered when an event message is raised by the Transaction Manager. Upon getting an event message, the Rule Manager models the *condition evaluation*. Satisfaction of a condition can lead to the triggering of one or

more of the immediate, deferred, and detached actions of the rule. Condition evaluation is performed probabilistically, using a separate probability value for each of the immediate, deferred, and detached coupling modes (see Table 1). A subtransaction corresponding to each triggered action is submitted to the Transaction Manager.

A detached subtransaction submitted to the Transaction Manager is treated as a new transaction and it is executed independent of the triggering transaction. Immediate and deferred subtransactions, on the other hand, are executed as a part of the parent transaction which has triggered them. They are associated with the same real-time priority as their parent. Some more sophisticated priority assignment policies for subtransactions were proposed in the literature [38,42]; however, those policies require some a priori knowledge about transactions like their estimated execution times. Our system does not assume the knowledge of execution time of transactions. The subtransactions might access remote as well as local data pages, and their access requests are scheduled in the same way as the parent transactions (according to the rules associated with one of the two transaction-processing architectures described in the next section). We assume that immediate and deferred subtransactions do not trigger further subtransactions; i.e., there is no cascading rule firings.

When an immediate subtransaction is submitted to the Transaction Manager, the execution of the parent transaction is suspended until the completion of the immediate subtransaction. All the deferred subtransactions triggered by a transaction are started to execute when their parent completes its operations. We assume that all the immediate and deferred subtransactions triggered by a transaction share the locks. Therefore, although the deferred subtransactions of a transaction are executed concurrently with the same priority, deadlock is still not possible with the High-Priority concurrency control protocol because the subtransactions do not block each other. When all the deferred subtransactions of a transaction complete their execution, the locks of the subtransactions and the parent transaction are released. Some of the assumptions of this active transaction model are relaxed in the nested transaction execution scheme described in Section 4.

2.2. Transaction processing architectures

We consider two different architectures for processing ARTDBS transactions: *distributed transaction* and *mobile data*. Both architectures described briefly in the following [3] assume that there exists exactly one copy of each data page in the system.

[3] Further details on the transaction-processing architectures can be found in Ref. [48].

Distributed transaction (*DT*) *architecture*: Each transaction 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. This architecture (also called *function shipping* or *database-call shipping*) was already studied for traditional distributed database management systems by a number of researchers (e.g., [7,20,31]). In our system, the priority of a transaction is carried by all of the cohorts of the transaction to be used in scheduling cohorts' executions. The Transaction Manager is responsible for the creation of the master process for each transaction. The master process coordinates the execution of cohorts through communicating with the Transaction Manager of each cohort's site. 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, one cohort will be executed locally. For each operation of the transaction a global data dictionary is referred 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, it is just activated 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), the High-Priority protocol is applied; i.e., if the lock-holding cohort has higher priority than the priority of the cohort that is requesting the lock, the latter cohort is blocked. Otherwise, the lock-holding cohort is aborted and the lock is granted to the high priority lock-requesting cohort.

For the atomic commitment of the distributed transactions, we use the centralized *two-phase commit* protocol [3]. The blocking delay of two-phase commit (i.e., the delay experienced at both the master process site and each of the cohort process sites while waiting for messages from each other) is explicitly simulated in conducting the performance experiments.

Mobile data (*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 it has been 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 the 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.

For each operation of a transaction T executed at site S_i, the data dictionary of S_i is referred to find out the origin of the required data page P. If page P has been originated at site S_i but currently being resided at another site, 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, the request message is forwarded to S_k.

Similar to DT, access to a data page is controlled on the basis of the High-Priority protocol. 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 [4]. If the lock is granted, the reply message contains both the grant and the requested page; otherwise, the message will cause 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.

It is ensured by this transaction-processing architecture that the current location of a data page can always be found out by communicating with the originating site of that page. 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, 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.

2.3. Configuration and workload parameters

The list of parameters described in Table 1 was used in specifying the configuration and workload of the distributed ARTDBS. It is assumed that each site has one CPU and one disk.

The time spent at each disk access is chosen uniformly from the range $0.5 * DiskAccessTime$ through $1.5 * DiskAccessTime$. The CPU spends *DiskOverheadInst* instructions for each I/O operation. A control message is a non-data message like commit, abort, lock-request, lock-grant messages etc., and the size of such messages is specified by *ControlMsgSize*.

The average transaction arrival rate at each of the sites is determined by the parameter *ArrivalRate*. Arrivals are assumed to be Poisson. The slack time of a transaction specifies the maximum length of time the transaction can be delayed and still satisfy its deadline. It is detailed in Appendix A how the Transaction Generator involves the parameter *SlackRate* in assigning deadline to transactions.

3. Performance experiments

The simulation program, capturing the details of the distributed ARTDBS model, was written in CSIM [39], which is a process-oriented simulation language based on the C programming language.

The default parameter values used in each of the 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 configuration and transaction parameters were basically taken from our earlier experiments [48]. It was intended by those settings to provide a transaction load and data contention high enough to bring out the differences between various alternative execution environments for distributed ARTDBSs. The default values used for the resource-related parameters can be accepted as reasonable approximations to what can be expected from today's systems.

The performance metric we used in our evaluations is *miss_ratio*, which determines the fraction of transactions that miss their deadlines. For each experiment, the final results were evaluated as averages over 20 independent runs. Each run continued until 1000 transactions were executed at each data site. 90% 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. Summary of results obtained for two different transaction-processing architectures without any triggering transactions

When we evaluated the transaction-processing architectures DT and MD in a distributed real-time database system environment without any active capabilities (i.e., in the absence of triggering transactions), we obtained the following results [48]. The relative performance of the architectures is primarily determined by the resource requirements of transactions processed under each of

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

Table 1
Distributed ARTDBS model parameters

Configuration parameters	
NrOfSites	Number of sites
DBSize	Size of the database in pages
MemSize	Number of pages that can be held in memory
PageSize	Page size in bytes
CPURate	Instruction rate of CPU at each site (MIPS)
DiskAccessTime	Average disk access time
DiskOverheadInst	CPU overhead for performing disk I/O
NetworkBandwidth	Network Bandwidth
ControlMsgSize	Control message size in bytes
FixedMsgInst	Fixed number of instructions to process a message
PerByteMsgInst	Additional number of instructions per message byte
Transaction parameters	
ArrivalRate	Average arrival rate of transactions at each site
TransSize	Average number of pages accessed by each transaction
SubTransSize	Average number of pages accessed by each subtransaction
RemoteAccessRate	Probability of accessing a page with a remote origin
StartTransInst	Number of instructions to initialize a transaction
EndTransInst	Number of instructions to terminate a transaction
ProcessPageInst	Number of CPU instructions to process a page
WriteProb	Probability of writing to a page
ImmediateProb	Probability of triggering an immediate subtransaction following a page update
DeferredProb	Probability of triggering a deferred subtransaction following a page update
DetachedProb	Probability of triggering a detached subtransaction following a page update
SlackRate	Average rate of slack time of a transaction to its processing time

the architectures. 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 and not the transmission time; and DT was characterized by the larger number of messages (compared to MD) issued for each transaction. When a fast network was used, on the other hand, the average volume of messages did not have much influence on the performance, and MD demonstrated superior performance. MD was also observed to produce less I/O delay in storing the updated pages on stable storage. With MD, the pages with remote origin that are updated by a transaction can be consecutively placed on the local disk preventing the delay of separate seek time for each stored page.

3.2. Results obtained with an active workload

3.2.1. Impact of transaction load

This experiment was conducted to observe the performance of the system under different levels of transaction load. Average transaction arrival rate at a site (i.e., *ArrivalRate*) was varied from 0.5 to 1.5 transactions per second in steps of 0.25. This range of *ArrivalRate* values corresponds to an average CPU utilization of about 0.93–0.55 at each data site.

It can be observed from Fig. 2 that as the transaction load is increased, more transactions miss their deadlines. Obviously, the increasing load leads to more data and resource conflicts among transactions, and therefore more blockings and priority aborts are experienced. The involvement of active workload does not change the general conclusions obtained for performance of the transaction-processing architectures. The discussion

Table 2
Distributed ARTDBS model parameter values

Configuration parameters	
NrOfSites	10
DBSize	2500 pages
MemSize	20% of the *DBSize*
PageSize	4096 bytes
CPURate	50 MIPS
DiskAccessTime	20 ms
DiskOverheadInst	5000 instructions
NetworkBandwidth	10 Mbps (e.g., Ethernet) or 100 Mbps (e.g., FDDI)
ControlMsgSize	256 bytes
FixedMsgInst	20,000 instructions
PerByteMsgInst	10,000 instructions per 4 Kbytes
Transaction parameters	
ArrivalRate	1 transaction per second
TransSize	10 pages
SubTransSize	4 pages
RemoteAccessRate	0.5
StartTransInst	30,000 instructions
EndTransInst	40,000 instructions
ProcessPageInst	30,000 instructions
WriteProb	0.5
ImmediateProb	0.5
DeferredProb	0.5
DetachedProb	0.5
SlackRate	10

provided in Section 3.1 for the relative performance results of architectures is applicable here as well. When a slow network is employed (i.e., *NetworkBandwidth* = 10 Mbps), the performance results obtained with DT and MD are comparable to each other. With a fast network (i.e., *NetworkBandwidth* = 100 Mbps), MD is the clear winner, especially under high levels of transaction load. DT experiences higher CPU delay due to processing larger number of messages and higher I/O delay in storing the updated pages on disk, as we discussed in the preceding section. The larger volume of messages experienced with MD does not have a considerable impact on the relative performance when a fast network is employed.

3.2.2. Impact of triggering probabilities

In this experiment, we examined the system behavior while the probability of triggering a subtransaction was varied for each of the three coupling modes: detached, immediate, and deferred. While evaluating the performance impact of a coupling mode, the parameter values used to determine the probabilities of the other coupling modes were kept constant at 0.5.

We first varied the probability of triggering a detached subtransaction following each page update (i.e., *DetachedProb*) from 0.0 to 1.0. The overall shapes of the curves presented in Fig. 3 remain the same as those in Fig. 2. Again, MD provides better performance with a fast network, and there is no considerable difference in performances of MD and DT with a slow network. The similarity of the performance results to those presented in the preceding section is not surprising as detached subtransactions are treated as new submissions (i.e., they are executed independent of their triggering transactions), and therefore increasing the number of detached subtransactions can be considered as another way of increasing the transaction load in the system. However, this argument is not applicable to the other coupling modes. As the subtransactions created in immediate or deferred modes are executed as a part of the triggering transaction, increasing the number of such subtransactions has an implication of increasing the average length of transactions, rather than directly increasing the transaction load. Fig. 4 displays the *miss_ratio* results obtained by varying the amount of immediate subtransactions. For the small number of triggered subtransactions (i.e., when the triggering probability is less than 0.5), the trends observed with varying probabilities of detached and immediate subtransactions are similar. However, when the system is characterized by a high volume of triggered subtransactions, an increase in the number of immediate subtransactions leads to a much sharper increase in the number of missed deadlines compared to that observed with the increase in the number of detached subtransactions. We contribute this result to the fact that extending the lifetime of transactions by involving some extra operations has more crucial effects on the real-time performance than executing those operations in the form of separate transactions. Fig. 5 presents the average number of data conflicts experienced by a transaction as a function of the probability of both immediate and detached coupling modes. The results for each coupling mode were obtained by setting the triggering probabilities of the other coupling modes to 0.5. The results presented were obtained with DT and a fast network. Similar trends were observed with the other possible combinations of the transaction-processing architecture and the network speed. A data conflict results in either transaction abort or blocking, and in any case the execution of a transaction is delayed. With both DT and MD architectures, increasing the number of immediate subtransactions has more adverse effects on the real-time performance compared to the effects of detached subtransactions; and this observation is

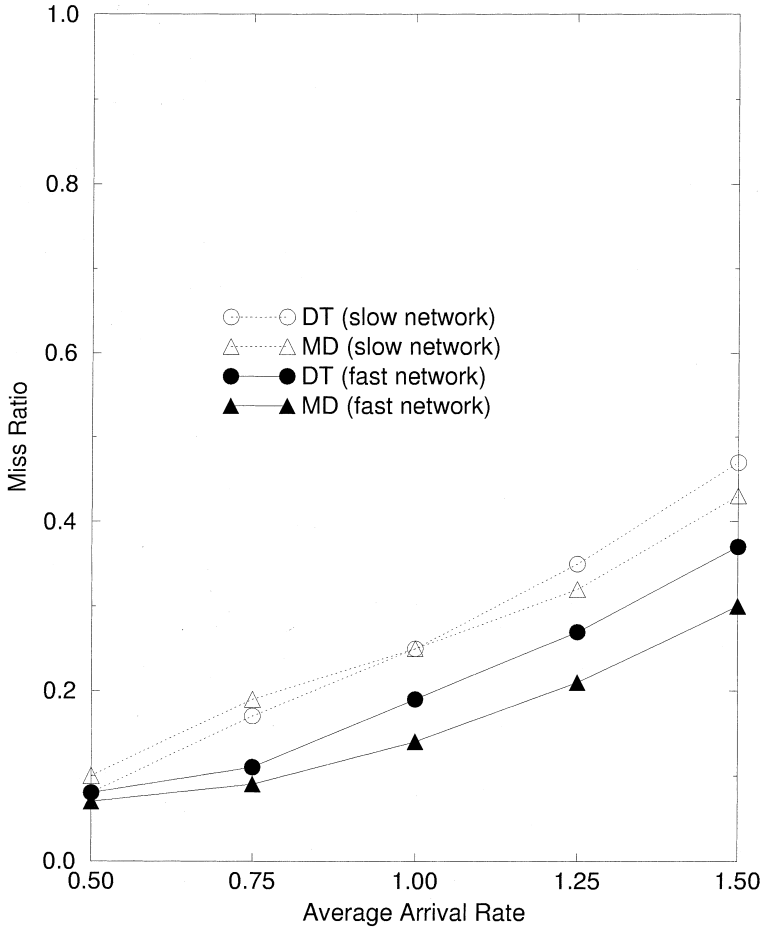

Fig. 2. Real-time performance in terms of the fraction of missed deadlines as a function of transaction load.

more apparent with DT architecture. The difference between the performances of DT and MD becomes more pronounced as the amount of immediate subtransactions increases. Data conflict aborts, that are experienced more with the immediate coupling mode, lead to much more message overhead with DT than that with MD. When a cohort of a transaction is aborted, DT architecture requires the master process of the transaction 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 should again communicate with the 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 does not require to generate new request messages.

The performance results we obtained by varying the probability of triggering deferred subtransactions on each data update are displayed in Fig. 6. Similar to immediate subtransactions, deferred subtransactions are also executed as a part of the triggering transaction. Therefore, the rapid increase in the number of missed deadlines is again a result of increasing the size of transactions by involving more deferred subtransactions. However, as a difference from the results obtained by varying the number of immediate subtransactions, we have a little bit better results for DT architecture. The performance results of DT and MD are closer to each other compared to those presented in Fig. 4. This result is due to the fact that while the deferred subtransactions of a transaction are started to execute at the end of the transactions, there is a chance with DT architecture to execute some of those subtransactions in parallel if they are accessing data pages stored at different sites.

The large number of deadline misses experienced with immediate and deferred subtransactions confirms the observation of Branding and Buchmann [5] that immediate and deferred coupling modes have some negative properties to be supported by real-time database systems.

3.2.3. Impact of subtransaction length

Fig. 7 provides the performance results obtained under different levels of data contention by varying the length of (i.e., number of data pages accessed by) each

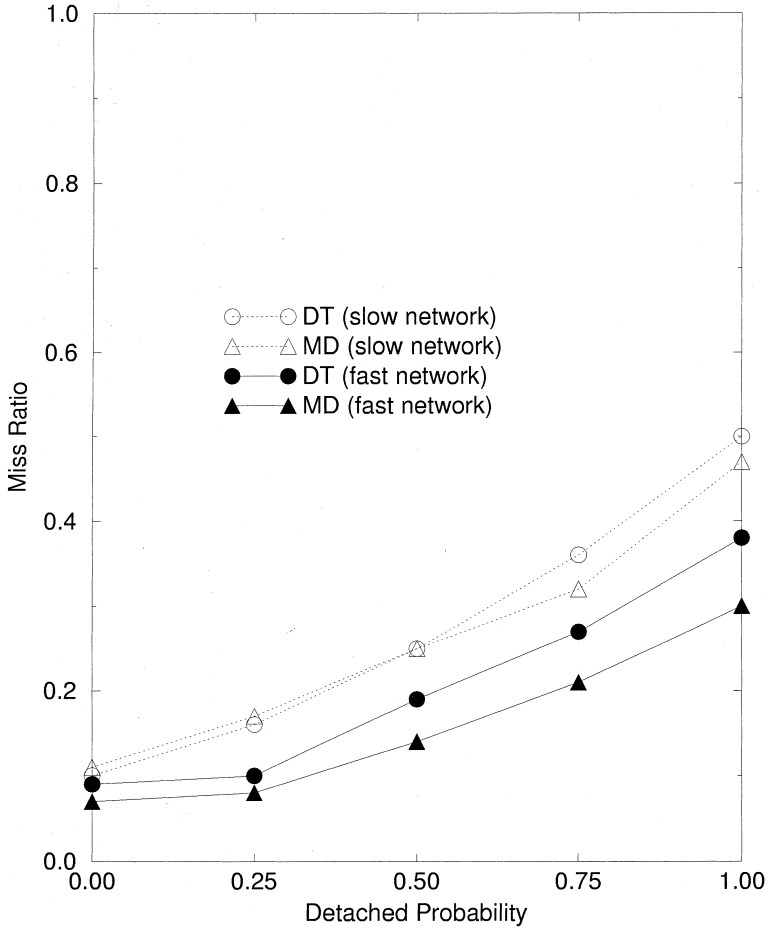

Fig. 3. Real-time performance in terms of the fraction of missed deadlines as a function of the probability of triggering detached subtransactions.

type of subtransactions. While conducting the experiment for different lengths of the subtransactions (i.e., *SubTransSize*) of a coupling mode, the length of subtransactions triggered in other coupling modes was set to 4 pages. The results were obtained by employing DT architecture with a fast network. For small lengths of subtransactions, the performance results obtained for different coupling modes are about the same. The difference between the performance results starts to appear when the length of a subtransaction triggered in a coupling mode is increased beyond 4. This difference becomes more pronounced with each additional data page accessed by a subtransaction. The real-time performance of the system is affected much more negatively when the size of immediate or deferred subtransactions is increased, compared to the performance degradation observed by increasing the detached subtransaction size. This result is due to, as explained in the preceding section, executing immediate and deferred subtransactions as a part of the triggering transaction. Data contention increases much faster when the lifetime of triggering transactions, which are larger than detached subtransactions, is extended further, compared to the increase in data contention due to increasing the size of detached subtransactions.

4. Adapting a nested transaction model

In this section we present the performance results obtained by adapting a nested transaction execution model to our ARTDBS. This model enables us to execute immediate subtransactions concurrently with their triggering transactions; in other words, a triggering transaction needs not to be suspended during the execution of its immediate subtransactions. In this case, we do not assume that immediate and deferred subtransactions triggered by a transaction share the locks. Again, a transaction and its subtransactions [5] are associated with the same priority, and a subtransaction does not trigger further subtransactions. The locking rules proposed by Harder and Rothermel [25] for nested transactions are adapted to our system as follows. Besides holding a lock, a

[5] For the remainder of this section, we use the term subtransaction to denote either immediate or deferred subtransaction.

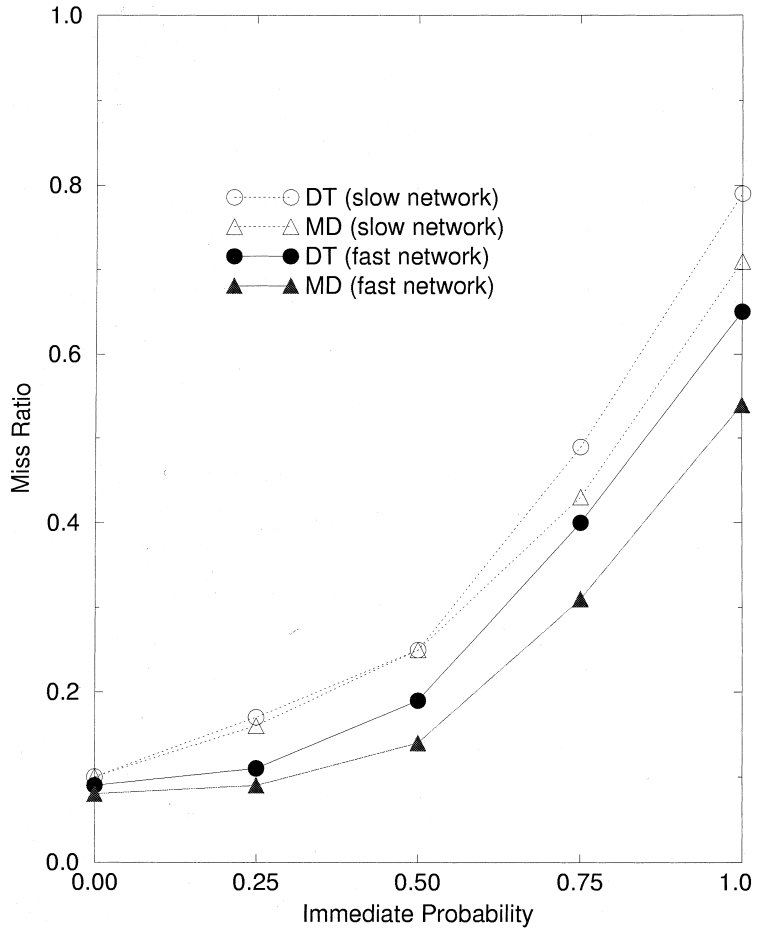

Fig. 4. Real-time performance in terms of the fraction of missed deadlines as a function of the probability of triggering immediate subtransactions.

transaction can *retain* a lock. When a subtransaction commits, the triggering parent transaction inherits the locks of the subtransaction and retains them. A retained lock does not give the right to its owner to access the locked page, rather it indicates that only the subtransactions of the retainer or the retainer itself can potentially acquire the lock.

When a subtransaction S requests a lock, the following protocol is executed:

if there exists any (sub)transaction S' that holds the lock
  if priority(S) > priority(S')
    S' is aborted;
    The lock is granted to S;
  else
    S is blocked;
else if there exists any transaction T that retains the lock and is not the parent of S
  if priority(S) > priority(T)
    T is aborted;
    The lock is granted to S;
  else
    S is blocked;
else
  The lock is granted to S;

The lock request of a transaction is handled similarly; the only difference is that the phrase "and is not the parent of S" should be replaced by "and is not the lock-requesting transaction itself".

When a subtransaction is aborted, its parent transaction is also aborted only if it retains at least one of the locks that have been held by the aborted subtransaction. All the locks that have been retained/held by an aborted (sub)transaction are released.

When we repeated the experiment that evaluates the performance impact of immediate subtransactions by involving the nested locking protocol described above, we observed some noticeable changes in the results. As displayed in Fig. 8, increasing the probability of triggering immediate subtransactions does not lead to as steep increase in missed deadlines as we can see in Fig. 4. Execution of immediate subtransactions and their triggering transactions in a concurrent manner can reduce the adverse effects of subtransactions on satisfying

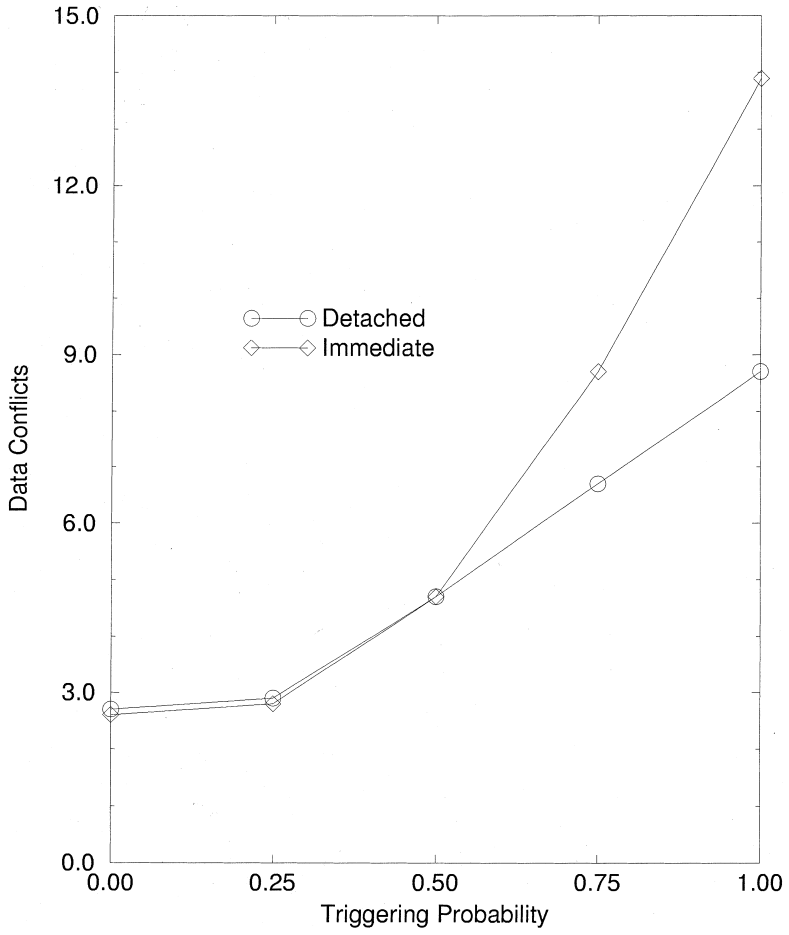

Fig. 5. Average number of data conflicts experienced by a transaction as a function of the probability of triggering detached/immediate subtransactions.

deadlines that we discussed in the preceding sections. Also, the nested execution model suffers much less from the overhead of priority aborts of subtransactions. A subtransaction abort does not always lead to the abort of the whole triggering transaction, as discussed above. Another difference from the previous performance results is that the transaction-processing architecture MD does not seem to be clearly preferable to DT in this case. The advantage of DT with the nested execution model is the possibility of executing the operations of a transaction and its immediate subtransactions in parallel if the operations require to access data pages at different sites. With MD, although a transaction and its subtransactions can be executed concurrently, their parallel execution is not possible because they are required to be executed together at their originating site. [6] The trend observed about the relative performance of MD and DT with the nested transaction model was confirmed by reconducting the transaction load experiment of Section 3.2.1. It is shown in Fig. 9 that the performance results of MD and DT are fairly close to each other even when a fast network is employed.

We also observed through another experiment that the nested execution scheme reduces the negative performance impact of deferred subtransactions as well. We do not display the results here as the performance trends are similar to those obtained for immediate subtransactions.

5. Conclusions

In this paper, by modeling the semantics of rule execution in the form of a transaction, we provided a performance analysis of various transaction execution strategies in a distributed active real-time database system (ARTDBS) environment. Using a detailed ARTDBS simulation model, we conducted a series of experiments to investigate various performance issues involved in processing distributed transactions. The performance metric we used in evaluations is the fraction of transactions that violate their timing constraints.

[6] Remember that, at each site we assume a uniprocessor system.

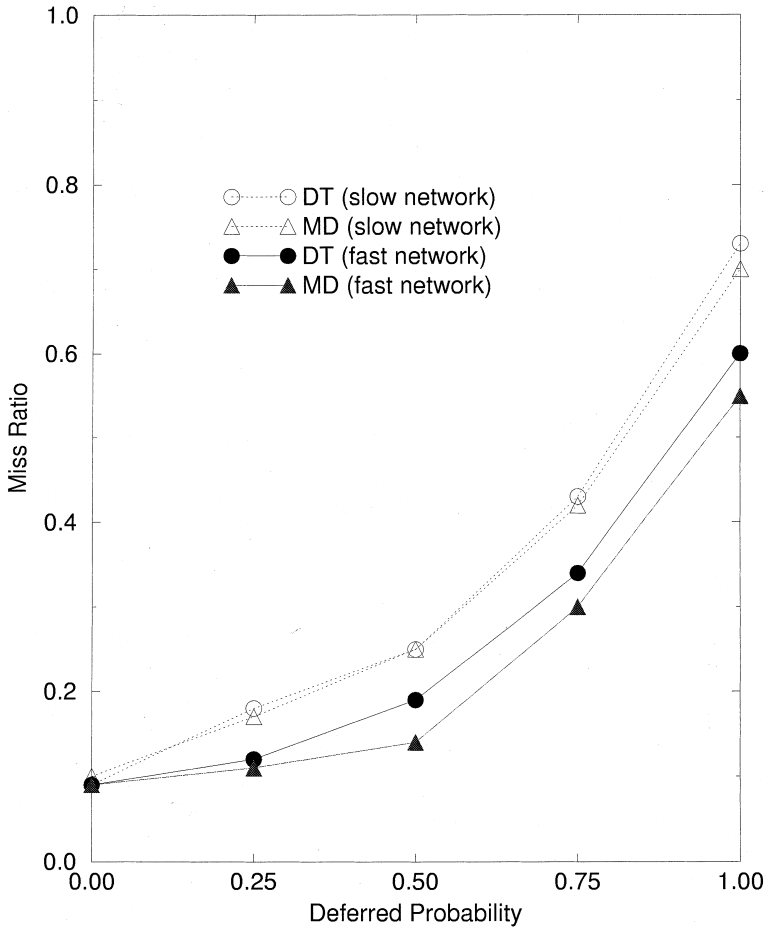

Fig. 6. Real-time performance in terms of the fraction of missed deadlines as a function of the probability of triggering deferred subtransactions.

Main results of our experiments can be summarized as follows. Increasing the probability of triggering immediate/deferred subtransactions led to a steep increase in the number of missed deadlines. The negative impact of detached subtransactions on the real-time performance was not that crucial. Since detached subtransactions are executed independent of their triggering transactions, increasing the number of detached subtransactions effectively corresponds to increasing the transaction load in the system. The direct effect of increasing the number of immediate or deferred subtransactions, on the other hand, is an increase in the average length of triggering transactions. Extending the lifetime of transactions by triggering such subtransactions was observed to result in much more data conflicts (and therefore more priority aborts and blockings) compared to increasing the transaction load by triggering detached subtransactions.

The performance results obtained with mobile data (MD) transaction-processing architecture were more satisfactory in general, compared to those obtained with distributed transaction (DT) architecture, under various types of active workload. When a fast network was employed, MD was observed to be the clear performance winner. Data conflict aborts lead to much more message overhead with DT than that with MD. The difference between the performances of DT and MD became more pronounced as the amount of immediate subtransactions was increased. However, the performances of DT and MD were comparable to each other under a high volume of deferred subtransactions. The deferred subtransactions triggered by a transaction have a chance to be executed in parallel with DT, if they access data pages stored at different sites.

Increasing the size of immediate and deferred subtransactions also caused a steep degradation in real-time performance due to rapidly increasing data and resource contention among larger-sized transactions.

In order to reduce the negative impact of immediate and deferred subtransactions on the real-time performance, we described a nested transaction execution model for the active workload. In that model, immediate and deferred subtransactions triggered by a transaction do not share the locks any more. The locks are managed on the basis of a nested locking protocol. Also, a triggering transaction needs not to be suspended during the execution of its immediate subtransactions. The results of the experiments involving the proposed nested execution

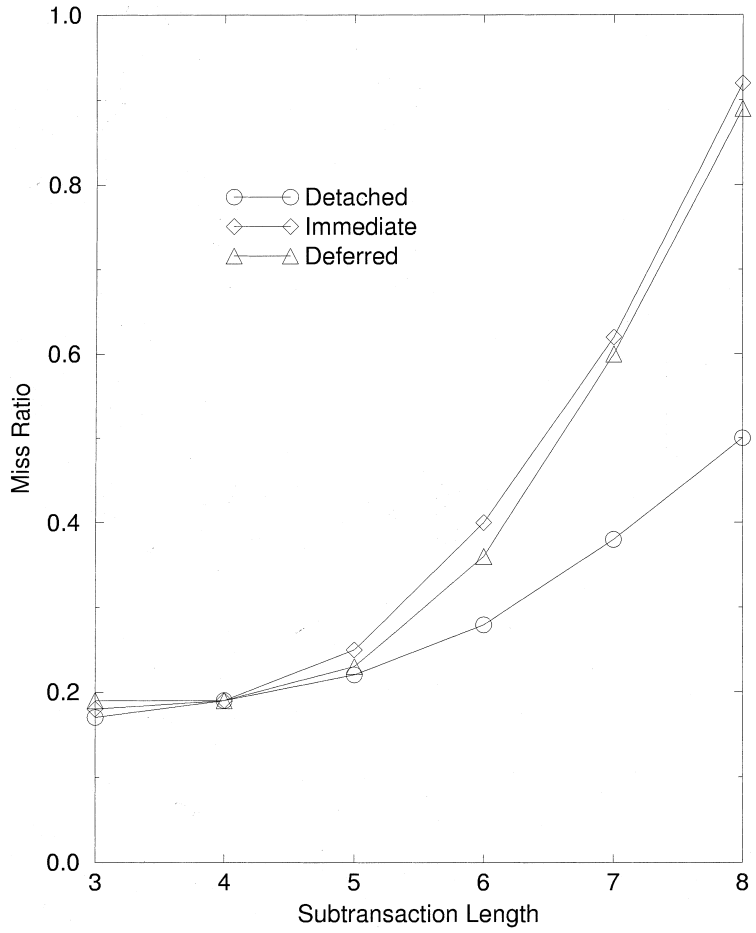

Fig. 7. Real-time performance in terms of the fraction of missed deadlines as a function of the length of a subtransaction triggered in a specific coupling mode.

model confirmed our intuition that the system suffers less with this model from the overhead of executing a high volume of triggered subtransactions. Providing a higher level of concurrency among immediate subtransactions and their parent, and reducing the overhead of subtransaction aborts played the major roles in improving the performance.

Acknowledgements

This research is supported by the Research Council of Turkey (TÜBİTAK) Grant EEEAG-246 and the NATO Collaborative Research Grant CRG 960648.

Appendix A

A.1. Deadline calculation

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 deadline D_T of a transaction T is determined by the following formula.

$$D_T = A_T + PT_T + S_T,$$

where

$$S_T = \exp(SlackRate * PT_T).$$

A_T, PT_T, and S_T denote the arrival time, estimated minimum processing time, and slack time of transaction T, respectively.

The estimated minimum processing time formula actually 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 that will be experienced by the transaction due to conflicts and remote accesses should not exceed the slack time included in the deadline formula.

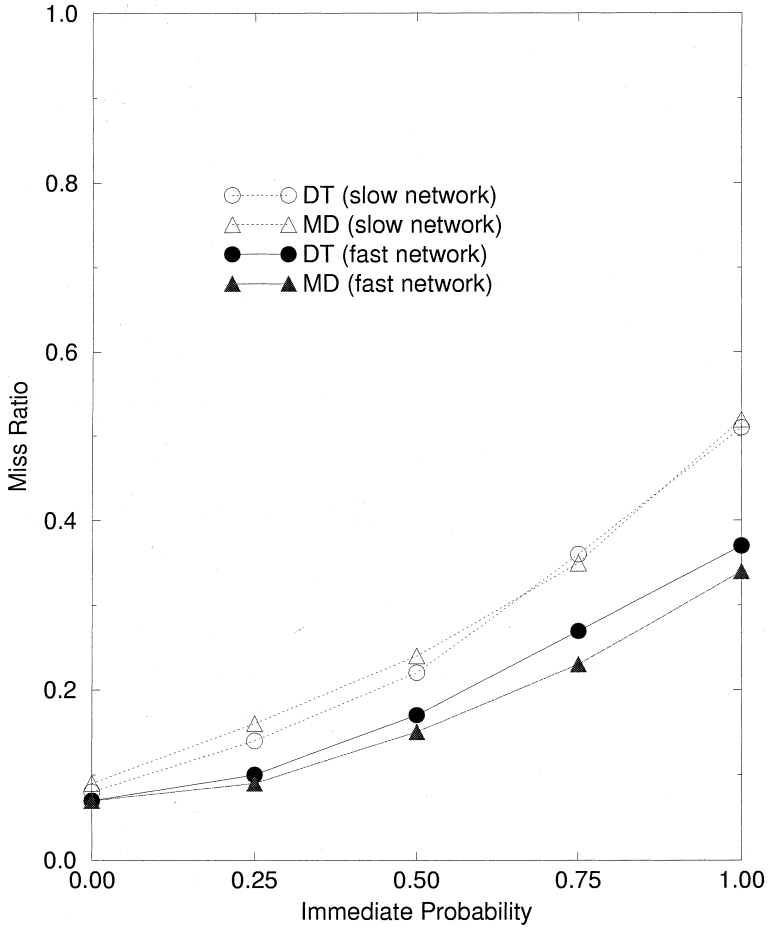

Fig. 8. Performance impact of immediate subtransactions when a nested transaction execution model is employed.

$$PT_T = PT_{T-\text{Parent}} + PT_{T-\text{Children}},$$

where $PT_{T-\text{Parent}}$ denotes the processing time spent due to the operations of T itself, and $PT_{T-\text{Children}}$ denotes the processing time due to the operations of immediate and deferred subtransactions triggered by T.

$$PT_{T-\text{Parent}} = CPU\_delay_{T-\text{Parent}} + I/O\_delay_{T-\text{Parent}},$$

$$CPU\_delay_{T-\text{Parent}} = \frac{10^{-3}}{CPURate} * (StartTransInst + (1 + WriteProb) * TransSize * ProcessPageInst + EndTransInst),$$

$$I/O\_delay_{T-\text{Parent}} = \left[\left(1 - \frac{MemSize}{DBSize}\right) * TransSize * \left(\frac{DiskOverheadInst}{CPURate} * 10^{-3} + DiskAccessTime\right)\right] + \left[WriteProb * TransSize * \left(\frac{DiskOverheadInst}{CPURate} * 10^{-3} + DiskAccessTime\right)\right].$$

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 $CPU\_delay_{T-\text{Parent}}$ and $I/O\_delay_{T-\text{Parent}}$ is milliseconds.

The computation of $PT_{T-\text{Children}}$ involves the probabilities of triggering immediate and deferred subtransactions.

$$PT_{T-Children} = TransSize * WriteProb * (ImmediateProb + DeferredProb) * (CPU\_delay_{T-\text{Children}} + I/O\_delay_{T-\text{Children}}).$$

The computations of $CPU\_delay_{T-\text{Children}}$ and $I/O\_delay_{T-\text{Children}}$ are similar to those of $CPU\_delay_{T-\text{Parent}}$ and $I/O\_delay_{T-\text{Parent}}$, respectively, except that *TransSize* is replaced by *SubTransSize* in both formulas.

In determining the deadline of a transaction, detached subtransactions that could be triggered by the transaction are not considered since such subtransactions are treated as independent transactions.

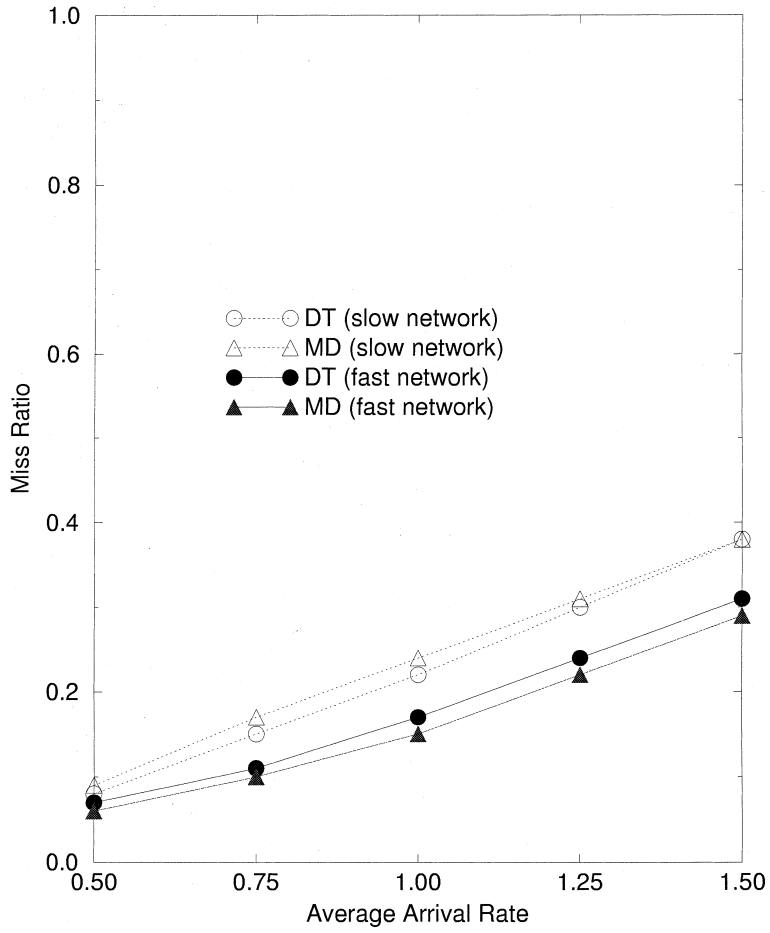

Fig. 9. Real-time performance as a function of transaction load when a nested transaction execution model is employed.

References

[1] R. Abbott, H. Garcia-Molina, Scheduling real-time transactions: A performance evaluation, ACM Transactions on Database Systems 17 (3) (1992) 513–560.

[2] M. Berndtsson, J. Hansson, Issues in active real-time databases, in: International Workshop on Active and Real-Time Database Systems, 1995, pp. 142–157.

[3] P.A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison–Wesley, Reading, MA, 1987.

[4] A. Bestavros, S. Braoudakis, Value-cognizant speculative concurrency control for real-time databases, Information Systems 21 (1996) 75–102.

[5] H. Branding, A. Buchmann, On providing soft and hard real-time capabilities in an active DBMS, in: International Workshop on Active and Real-Time Database Systems, 1995, pp. 158–169.

[6] A. Buchmann, J. Zimmermann, J. Blakeley, D.L. Wells, Building an integrated active OODBMS: Requirements, architecture and design decisions, in: International Conference on Data Engineering, 1995, pp. 117–128.

[7] M.J. Carey, M. Livny, Distributed concurrency control performance: A study of algorithms, distribution, and replication, in: International Conference on Very Large Data Bases, 1988, pp. 13–25.

[8] M.J. Carey, R. Jauhari, M. Livny, On transaction boundaries in active databases: A performance perspective, IEEE Transactions on Knowledge and Data Engineering 3 (3) (1991) 320–336.

[9] S. Chakravarthy, HiPAC: A research project in active, time-constrained database management, Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge (1989).

[10] S.Chakravarthy, V. Krishnaprasad, E. Abwar, S.K. Kim, Anatomy of a composite event detector, Technical Report UF-CIS-TR-93-039, CIS Department, University of Florida (1993).

[11] M. Chen, K.J. Lin, Dynamic priority ceilings: A concurrency control protocol for real-time systems, Real-Time Systems 2 (4) (1990) 325–346.

[12] Y.W. Chen, L. Gruenwald, Effects of deadline propagation on scheduling nested transactions in distributed real-time database systems, Information Systems 21 (1) (1996) 103–124.

[13] A. Datta, S.H. Son, A study of concurrency control in real-time active database systems, Technical Report, Department of MIS, University of Arizona (1996).

[14] A. Datta et al., Multiclass transaction scheduling and overload management in firm real-time database systems, Information Systems 21 (1) (1996) 29–54.

[15] U. Dayal et al., The HiPAC project: Combining active database and timing constraints, ACM SIGMOD Record 17 (1) (1988) 51–70.

[16] U. Dayal, Active database management systems, in: Third International Conference on Data and Knowledge Bases, 1988, pp. 150–169.

[17] U. Dayal, M. Hsu, R. Ladin, Organizing long-running activities with triggers and transactions, in: ACM SIGMOD Conference, 1990.

[18] O. Diaz, N. Paton, P. Gray, Rule management in object-oriented databases: A uniform approach, in: International Conference on Very Large Data Bases, 1991, pp. 317–326.
[19] L.C. DiPippo, V.F. Wolfe, Object-based semantic real-time concurrency control, in: Real-Time Systems Symposium, 1993, pp. 87–96.
[20] H. Garcia-Molina, R.K. Abbott, Reliable distributed database management, Proceedings of the IEEE 75 (1987) 601–620.
[21] S. Gatziu, K.R. Dittrich, Events in an active object-oriented database system, in: International Workshop on Rules in Database Systems, 1993.
[22] S. Gatziu, K.R. Dittrich, Detecting composite events in active database systems using petri nets, in: International Workshop on Research Issues in Data Engineering, 1994.
[23] N. Gehani, H.V. Jagadish, O. Shumeli, Composite event specification in active databases: Model and implementation, in: International Conference on Very Large Data Bases, 1992.
[24] R. Gupta, J. Haritsa, K. Ramamritham, S. Seshadri, Commit processing in distributed real-time database systems, in: IEEE Real-Time Systems Symposium, 1996.
[25] T. Harder, K. Rothermel, Concurrency control issues in nested transactions, VLDB Journal 2 (1) (1993) 39–74.
[26] J.R. Haritsa, M.J. Carey, M. Livny, Data access scheduling in firm real-time database systems, Real-Time Systems 4 (3) (1992) 203–241.
[27] D. Hong, T. Johnson, S. Chakravarthy, Real-time transaction scheduling: A cost conscious approach, in: ACM SIGMOD Conference, 1993, pp. 197–206.
[28] J. Huang, J.A. Stankovic, K. Ramamritham, D. Towsley, B. Purimetla, Priority inheritance in soft real-time databases, Real-Time Systems 4 (3) (1992) 243–268.
[29] M. Hsu, R. Ladin, D. McCarthy, An execution model for active database management systems, in: International Conference on Data and Knowledge Bases, 1988, pp. 171–179.
[30] B. Kao, H. Garcia-Molina, Deadline assignment in a distributed soft real-time system, in: International Conference on Distributed Computing Systems, 1993, pp. 428–437.
[31] W.H. Kohler, B.H. Jeng, Performance evaluation of integrated concurrency control and recovery algorithms using a distributed transaction testbed, in: International Conference on Distributed Computing Systems, 1986, pp. 130–139.
[32] K.Y. Lam, S.L. Hung, Concurrency control for time-constrained transactions in distributed database systems, Comput. J. 38 (9) (1995) 704–716.
[33] K.Y. Lam, V.C.S. Lee, S.L. Hung, B.C.M. Kao, Priority assignment in distributed real-time databases using optimistic concurrency control, International workshop on Real-Time Computing Systems and Applications, 1996, pp. 128–135.
[34] K.Y. Lam, J. Cao, C.L. Pang, S.H. Son, Resolving conflicts with committing transactions in distributed real-time databases, in: IEEE International Conference on Engineering of Complex Computer Systems, 1997, pp. 49–58.
[35] J. Lee, S.H. Son, Concurrency control algorithms for real-time database systems, in: Performance of Concurrency Control Algorithms in Centralized Database Systems, Prentice-Hall, Englewood Cliffs, NJ, 1995.
[36] G. Özsoyoğlu, R.T. Snodgrass, Temporal and real-time databases: A survey, IEEE Transactions on Knowledge and Data Engineering 7 (4) (1995) 513–532.
[37] N.M. Paton, O. Diaz, Active database systems, Technical Report, Heriot-Watt University, Edinburgh, Scotland (1994).
[38] B. Purimetla, R.M. Sivasankaran, J.A. Stankovic, K. Ramamritham, D. Towsley, Priority assignment in real-time active databases, in: Conference on Parallel and Distributed Information Systems, 1994.
[39] H. Schwetman, CSIM: A C-based, process-oriented simulation language, in: Winter Simulation Conference, 1986, pp. 387–396.
[40] L. Sha, R. Rajkumar, S.H. Son, C.H. Chang, A real-time locking protocol, IEEE Transactions on Computers 40 (7) (1991) 793–800.
[41] R.M. Sivasankaran, K. Ramamritham, J.A. Stankovic, D. Towsley, Data placement, logging and recovery in real-time active databases, in: International Workshop on Active and Real-Time Database Systems, 1995, pp. 226–241.
[42] R. Sivasankaran, J.A. Stankovic, D. Towsley, B. Purimetla, K. Ramamritham, Priority assignment in real-time active databases, The VLDB Journal 5 (1996) 19–34.
[43] S.H. Son, C.H. Chang, Performance evaluation of Real-Time locking protocols using a distributed software prototyping environment, in: International Conference on Distributed Computing Systems, 1990, pp. 124–131.
[44] N. Soparkar, E. Levy, H.F. Korth, A. Silberschatz, Adaptive commitment for real-time distributed transactions, TR-92-15, Department of Computer Science, University of Texas at Austin (1992).
[45] Ö. Ulusoy, G.G. Belford, Real-time lock based concurrency control in a distributed database system, in: International Conference on Distributed Computing Systems, 1992, pp. 136–143.
[46] Ö. Ulusoy, G.G. Belford, Real-time transaction scheduling in database systems, Information Systems 18 (8) (1993) 559–580.
[47] Ö. Ulusoy, Processing real-time transactions in a replicated database system, Distributed and Parallel Databases 2 (4) (1994) 405–436.
[48] Ö. Ulusoy, A study of two transaction processing architectures for distributed real-time database systems, Journal of Systems and Software 31 (2) (1995) 97–108.
[49] Ö. Ulusoy, Research issues in real-time database systems, Information Sciences 87 (1) (1995) 123–151.

Özgür Ulusoy received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1992. He is currently on the faculty of the Computer Engineering and Information Science Department at Bilkent University. His research interests include real-time database systems, active database systems, real-time communication, and mobile database systems. Dr. Ulusoy has served on numerous program committees for conferences and edited a Special Issue on Real-Time Databases in Information Systems Journal. He has published over 30 articles in archived journals and conference proceedings.