NORTH-HOLLAND

# Research Issues in Real-Time Database Systems

*Survey Paper*

ÖZGÜR ULUSOY

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

ABSTRACT

Today's real-time systems are characterized by managing large volumes of data. Efficient database management algorithms for accessing and manipulating data are required to satisfy timing constraints of supported applications. Real-time database systems involve a new research area investigating possible ways of applying database systems technology to real-time systems. Management of real-time information through a database system requires the integration of concepts from both real-time systems and database systems. Some new criteria need to be developed to involve timing constraints of real-time applications in many database systems design issues, such as transaction/query processing, data buffering, CPU, and IO scheduling. In this paper, a basic understanding of the issues in real-time database systems is provided and the research efforts in this area are introduced. Different approaches to various problems of real-time database systems are briefly described, and possible future research directions are discussed.

## 1. INTRODUCTION

There has recently been a great deal of interest in applying database technology to the management of data in real-time systems. This has resulted in the emergence of a new research area, called real-time database systems (RTDBSs). RTDBSs have inherited many properties from both real-time systems and database systems. Similar to a conventional real-time system, transactions processed in an RTDBS are associated with timing constraints, usually in the form of deadlines. Access requests of transactions to data or other system resources are scheduled on the basis of the timing constraints. What makes an RTDBS different from a real-time

0020-0255/95/$9.50
SSDI 0020-0255(95)00130-H

system is the requirement of preserving the logical consistency of data in addition to considering the timing constraints of transactions. The requirement of maintaining data consistency is the essential feature of a conventional database system. However, the techniques used to preserve data consistency in database systems are all based on transaction blocking and transaction restart, which makes it virtually impossible to predict computation times and hence to provide schedules that guarantee deadlines in an RTDBS. As a result, it becomes necessary to extend traditional database management techniques with time-critical scheduling methods. While the basic scheduling goal in a conventional database system is to minimize the response time of transactions and to maximize throughput, an RTDBS scheduler primarily aims to maximize the number of transactions that satisfy their deadlines.

RTDBSs entered the computer science spotlight with the publication of a Special Issue of the ACM SIGMOD Record [58] in 1988. The papers in that issue described the role of database systems in real-time applications and introduced some inspiring concepts. It was pointed out by the authors that effective and efficient methods are necessary for the management of large volumes of data maintained by real-time systems. Since then, the results of a considerable number of works addressing various features of RTDBSs have appeared in the literature.

The goals of this paper are to provide a basic understanding of the issues in RTDBSs, to introduce the research efforts in this area, and to suggest directions for future work. We organize the paper as follows. The next section explores the issues related to transaction scheduling in RTDBSs. It provides an examination of techniques used in mapping timing constraints into priorities, and describes priority-driven algorithms proposed for IO scheduling, buffer management, and concurrency control. Section 3 provides a brief description of approaches addressing various issues of "distributed" RTDBSs. Recent research efforts to integrate active database systems with RTDBSs are discussed in Section 4. Some architectural considerations to obtain better performance in RTDBSs are described in Section 5. The final section provides a brief summary of concepts and possible future research directions.

## 2. SCHEDULING IN REAL-TIME DATABASE SYSTEMS

Although it seems essential for many time-critical applications to combine scheduling methods from both real-time systems and database systems, this combination is not an easy task because of the distinct features of these two systems. Ramamritham [52] discusses the characteristics of real-time systems and database systems that are relevant to RTDBSs. Data handled by real-time systems are usually characterized as being *temporal*; i.e., data value is valid (up-to-date) only for a certain length of time. To quantify the notion of temporal data, each data item can be associated with a *valid interval* [60]. The actual state of the environment can only be presented during the valid interval of data. *Temporal consistency* can be achieved only if data items are accessed within their valid intervals. The temporal consistency requirement of data together with the fast response time requirements of the supported application establishes timing constraints for the transactions processed in the system. The primary scheduling goal in real-time systems is to satisfy the timing constraints of transactions.

Traditional database systems, on the other hand, more usually maintain *persistent* data. Transactions retrieving or updating shared data are required to preserve the logical consistency of the data (i.e., they must execute in a logically correct manner). Typically, no timing constraints are associated with transactions. The basic performance goal, in this case, is to maximize throughput or to minimize the average response time of transactions.

An RTDBS requires an integrated approach to consider data consistency requirements and timing constraints together in scheduling transactions. The remainder of this section provides a review of the recent work on various aspects of transaction scheduling in RTDBSs.

## 2.1. *PRIORITY ASSIGNMENT*

A transaction $T$ processed in an RTDBS is associated with the following attributes:

- $A_T$: Arrival time of $T$.
- $D_T$: Deadline of $T$.
- $S_T$: Slack time[1] of $T$.
- $E_T$: Execution time of $T$.
- $V_T$: Value[2] of $T$.
- $P_T$: Priority of $T$.

[1]The *slack time* of a transaction is defined as the maximum length of time the transaction can be delayed and still satisfy its deadline.

[2]As will be discussed shortly, the value represents the importance of a transaction.

The first four attributes are related by

$$D_T = A_T + E_T + S_T. \tag{1}$$

The *deadline* of a transaction indicates that it is required to complete the transaction before a certain time in the future. A typical categorization of transactions concerns the strictness of the deadlines assigned.

- *Hard deadline transactions* are associated with strict deadlines and the correctness of transaction operations depends on the time at which the results are produced [65]. The system must provide schedules that guarantee deadlines. Nuclear power plants, air traffic control systems, process control systems, and robotics are some examples of applications that usually process hard deadline transactions.
- *Soft deadline transactions* are scheduled based on their deadlines, and satisfaction of deadlines is still the primary performance goal in scheduling transactions; however, in this case, there is no guarantee that all deadlines will be met. A soft deadline transaction is executed until completion regardless of whether its deadline has expired or not.
- *Firm deadline transactions* also do not carry strict deadlines, i.e., missing a deadline may not result in a catastrophe, but unlike soft deadline transactions, they are aborted by the system once their deadlines expire. Typically, no value will be imparted to the system if a firm deadline transaction misses its deadline.

Real-world examples of applications supporting soft or firm deadline transactions are provided in [4]. Banking systems and airline-reservation systems usually process soft deadline transactions. When a customer submits a transaction, if the system cannot generate a response to the transaction within its deadline, the customer prefers getting the response late to not getting it at all. Stock market trading is an example of applications supporting firm deadline transactions. If, for instance, a transaction is submitted to learn the current price of a particular stock, the system should either return the result in a specified time period or not perform the operation at all, because conditions in the stock market can change very fast.

As stated before, the basic scheduling goal in a real-time application environment is to meet transaction deadlines. The scheduler thus assigns a priority to each transaction based on its deadline. Two of the most popular priority assignment schemes based on transaction deadlines are as follows:

- *Earliest Deadline First* (*EDF*): A transaction with an earlier deadline has higher priority than a transaction with a later deadline.

- *Least Slack First* (*LSF*): The transaction with the least slack time has the highest priority. When a transaction $T$ arrives at the system, its slack time $S_T$ can be evaluated using the other attributes of $T$ [see equation (1)]:

  $$S_T = D_T - (A_T + E_T).$$

  The dynamic version of the LSF deadline assignment scheme requires the evaluation of transaction priorities at each decision point [36]. Let $PT_T(t)$ and $S_T(t)$ denote the processing time spent so far by $T$ and the slack time of $T$ at time $t$, respectively. The slack time of $T$ at decision point $t$ can be determined by the following formula:

  $$S_T(t) = D_T - (t + E_T - PT_T(t))$$

Abbott and Garcia-Molina [1, 4] evaluated the performance of these priority assignment methods in an RTDBS. They observed that the EDF scheme leads to better performance (i.e., fewer missed deadlines) under light or moderate levels of concurrent transaction load. The schemes were shown to perform the same in RTDBSs characterized by high loads of transaction.

In a more recent work, Pang et al. [50] attempted to evaluate the performance of the EDF scheme on RTDBSs with multiclass workloads, where classes are distinguished by their transaction sizes. It was shown in that work that EDF discriminates significantly against longer transactions in attempting to minimize the number of late transactions. To overcome that bias, they introduced a dynamic priority assignment policy, called *Adaptive Earliest Virtual Deadline*, which attempts to ensure that long transactions are allocated a fair share of the system resources. This policy uses a sequence of *virtual deadlines* for a transaction to control the pace at which the transaction progresses toward meeting its deadline. It divides the transactions into a "hit" group and a "miss" group. Transactions in the hit group are given preferential access to resources to enhance the chances that they will make their deadlines. The virtual deadlines assigned to the transactions in the hit group are adjusted dynamically as the transactions progress, and a transaction with an earlier virtual deadline is served before one with a later virtual deadline. To overcome EDF's discriminatory behavior, the progress of longer transactions are monitored more closely; i.e., their virtual deadlines are adjusted more frequently.

Some applications may assign different values to transactions, where the *value* of a transaction reflects the return the application expects to receive if the transaction is completed before its deadline [31]. The scheduling goal for such applications is to maximize the value realized by the completed transactions. Biyabani et al. [8], Huang et al. [31], and Haritsa et al. [27] discuss some methods to establish a priority ordering among transactions that are distinguished by both values and deadlines. A key point considered in all those works is that value and deadline are two independent characteristics of RTDBS transactions. A close deadline does not necessarily imply a high value. Transactions with the same value may have different deadlines, while transactions with the same deadline may have different values.

A number of priority assignment algorithms considering both the values and deadlines of transactions have been proposed. A range of trade-offs between value and deadline has been covered in those algorithms. One common algorithm gives equal weight to deadline and value in determining the priority of transactions. The priority of transaction $T$ is specified by $P_T = V_T / D_T$. A variation of this algorithm uses the relative deadline instead of the absolute deadline in assigning priorities. The relative deadline is defined as the difference of the transaction deadline and the transaction arrival time; i.e., $P_T = V_T / (D_T - A_T)$. Haritsa et al. [27] introduced a bucket algorithm that allows the trade-off between transaction value and transaction deadline to be varied. The actual trade-off made between values and deadlines is controlled by a parameter of the algorithm.

In evaluating the system performance under various priority assignment policies, different performance metrics were considered by different researchers. The metric used in [8] combines the performance measurements of all value classes in terms of the fraction of guaranteed deadlines. The results presented in that work reveals that giving higher weight to deadline than to value in determining priorities results in better performance at low transaction loads. However, the situation is reversed under high levels of load; i.e., value should be given higher weight. The transactions considered in [31] are associated with soft deadlines; i.e., there is still some (but diminishing) value for completing the transactions after their deadlines. The primary metric involved in evaluations is the total value realized by all transactions processed in the system. It is shown in that work that considering values and deadlines together in assigning priorities provides a substantial improvement in performance compared to policies that do not combine those two attributes in formulating the priority. Another observation is that both the value and deadline distributions strongly affect the performance. The performance metric used in [27] is the total value provided by transactions that complete before their deadlines. The transaction deadlines are considered to be firm; i.e., no value is realized if the deadline is missed. The bucket algorithm proposed by the authors to assign priorities was fond to perform well under all operating conditions when its parameter is set appropriately. A performance improvement is provided by this algorithm over the other priority assignment policies discussed above.

### 2.2. *IO SCHEDULING*

In conventional database systems, the time spent for disk IO has been characterized as a dominant factor in overall system behavior. While modern microprocessor technology is advancing at an incredible rate (speedups of 40 to 60 percent annually), performance improvements in disk units are occurring at only about 7 to 10 percent annually [53]. As a result, just like in conventional systems, an important candidate for performance improvement in disk-resident RTDBSs is the IO subsystem. A conventional disk scheduling algorithm such as the *Shortest Seek Time First* or *SCAN* (*Elevator*) orders the sequence of IO requests to minimize the average disk head seek time [17]. On the other hand, the disk scheduler in an RTDBS primarily concerns the timing constraints of transactions in processing data access request [3, 11, 14, 39].

As discussed in [52], one important issue in scheduling the IO requests of an RTDBS transaction is the assignment of individual deadlines to the requests on the basis of the transaction deadline. This issue has not been addressed explicitly by the recent work performed on IO scheduling. We suggest that this problem can be seen as a different version of the serial subtask deadline assignment problem studied in [37] (see Section 3.3). The methods provided in that work can also be used to assign individual deadlines to IO requests.

Abbott and Garcia-Molina [3] developed some variants of the traditional SCAN algorithm in order to meet the deadlines of individual requests. Using simulation, one of the new algorithms, called *FD-SCAN*, was shown to consistently have the best performance in a wide variety of experiments. In this algorithm, the request with the earliest feasible deadline is chosen as the target and determines the scanning direction. (A deadline is feasible if it is estimated that it can be met.) If there is no request with a feasible deadline, then simply the closest request is serviced.

In [2], Abbott and Garcia-Molina studied an IO architecture that handles read and write request differently. The architecture assumes that write operations of a transaction are always performed after the transaction has committed. While a read request is assigned a priority based on the timing constraint of the transaction that issued it, a write request is assumed to have no explicit timing constraint; i.e., it is assigned the lowest priority. The rationale behind this policy is that giving high priority to a write does not enhance the performance directly, since the transaction that issued the write has already committed. An extension to this policy was provided by Kim and Srivastava [39] on the basis of the priority inheritance[3] rule. Assuming that a write lock is held until after the modified data item is copied into the database, if a transaction is waiting for the release of a write lock, the write request inherits the priority of the waiting transaction to activate the transaction as soon as possible. Otherwise (if no transaction is waiting), the write request gets the lowest priority among all the IO requests in the queue.

Carey et al. [11] proposed a priority-based variant of the SCAN algorithm. In this algorithm, disk requests are grouped on the basis of their priority, and the requests in each group are ordered using the traditional SCAN algorithm. On the completion of each request, if a disk request of a higher priority is found waiting to be serviced, the scheduler switches to service the requests in the higher priority group. The results of a simulation study provided by the authors indicate that the proposed algorithm is effective if it is used in conjunction with a priority-based buffer management algorithm.

Although using the priority-based disk scheduling algorithms described above helps IO requests meet their timing constraints, the overall IO performance in terms of the average seek time can become worse since some requests can receive very poor service. The algorithms provided in [14] attempt to reduce the overall seek time while taking the timing constraints into account in servicing the requests. In assigning request priorities, both the location and the deadline of requests are considered. A request very close to the disk arm can be assigned a high priority even if it has a large deadline. The algorithms were shown to perform well in terms of both the average seek time and the fraction of satisfied timing constraints.

## *2.3. BUFFER/MEMORY MANAGEMENT*

The problem of priority scheduling at the buffers of a database management system was first addressed by Carey et al. [11]. In that work, the variants of two existing buffer management algorithms that include priority considerations in buffer management decisions were presented. The first algorithm, called *Priority-LRU*, is the prioritized version of the *Global-LRU*

[3]The priority inheritance method is discussed more extensively in the context of concurrency control in Section 2.4.

algorithm. The algorithm organizes the buffer pool into priority levels, where each level consists of pages whose owners have the same priority. The pages within each level are arranged in the LRU order. When a page needs to be replaced, the least recently used page of the lowest priority is chosen as the victim. The second algorithm, called *Priority-DBMIN*, is an extension of the *DBMIN* buffer management policy [15]. In this policy, a set of buffers, called a "locality set," is allocated to each transaction for each file accessed by it. An optimizer provides the optimum size of each locality set and the optimum replacement policy to be used within each locality set. The priority-DBMIN algorithm allows a transaction to enter the system only if its optimally sized locality sets can be allocated. Otherwise, the lower priority transactions in the system are suspended until sufficient buffers become available for the new transaction. Using simulation, both algorithms were shown to be effective in enabling the system to achieve its performance goals. It was also shown that Priority-DBMIN dominates Priority-LRU in cases where buffer contention is a factor.

Jauhari et al. [35] suggest that although Priority-DBMIN was shown to outperform Priority-LRU, it is more difficult to implement due to the overhead of added system complexity. They proposed an easier-to-implement priority-based buffer management algorithm, called *Priority Hints*, and conducted simulation experiments to explore its performance. In the proposed algorithm, all the buffers owned by a transaction are organized into a "transaction set." Transaction sets are arranged in priority order. Two types of buffers can exist in a transaction set: the buffers containing fixed (i.e., currently being processed) pages, and the buffers containing unfixed favored (i.e., likely to be accessed) pages. When a replacement is required, nonfavored pages are considered first. If no nonfavored page exists, the most recently unfixed favored page of the lowest-priority transaction is chosen as the replacement victim. The idea of maintaining the favored pages in the most recently used (MRU) order is based on a discussion provided in [15], which states that MRU is a better approach than LRU when choosing replacement victims from a set of favored pages that are being repeatedly accessed. Performance experiments provided in [35] show that buffer management can have a very significant effect on the performance of a priority-oriented database system. For most workloads, the proposed algorithm (Priority Hints) was shown to perform as well as Priority-DBMIN, and better than Priority-LRU.

Abbott and Garcia-Molina [3] presented and evaluated two new buffer management techniques to be used in scheduling IO requests with deadlines. On the basis of the IO architecture they provided in [2] (as discussed in the preceding section), read and write requests are treated separately by the proposed techniques. Read requests are assumed to be issued by uncommitted transactions and receive service in accordance with the timing constraints of the transactions that issued them. Write requests, on the other hand, do not have explicit timing constraints because they are processed after the commitment of transactions. Read requests are buffered in a separate queue from write requests. The first buffer management policy proposed, called *Space Threshold* maintains a minimum amount of free space in the write buffer at all times (the amount is determined by the threshold parameter of the policy), so that each new write request can be placed in the buffer. Read requests are always preferred to write requests as long as the threshold is not exceeded. A write request is serviced only if the space threshold has been exceeded or there exists no read request in the system. The second buffer management policy, called *Time Threshold*, creates an artificial deadline $D_W$ for the action of writing the contents of a buffer slot to disk. The strictness of $D_W$ reflects the urgency of emptying a buffer slot; i.e., $D_W$ gets closer as a greater portion of the buffer becomes full. A write request is serviced only if $D_W$ is smaller than the earliest read deadline or there are no read requests. Through simulation, both Space Threshold and Write Threshold methods were shown to be effective in meeting read deadlines [3].

Pang et al. [51] presented a memory management algorithm to schedule real-time queries that require large amounts of computational memory (e.g., external sorting or join algorithms). The algorithm provides admission control and memory allocation of queries based on their timing constraints. The number of queries that can be admitted to memory at any time is controlled by dynamically choosing a target multiprogramming level to balance the demands on the system's memory, CPU, and disks. Then, the amount of memory assigned to each of the admitted queries is determined. One of the following two strategies is employed in memory allocation: the *Max* strategy, which assigns to each query either its maximum required memory or no memory at all, and the *MinMax* strategy, which assigns to low-priority queries their minimum required memory and to high-priority ones their maximum requirements. The workload characteristics of the system are considered in choosing one of these two strategies. The performance of the algorithm was studied using simulation under query workloads that perform hash joins or external sorts. The algorithm was shown to work well under overload situations and fluctuating workloads.

## *2.4. CONCURRENCY CONTROL*

*Concurrency control* in database systems is used to control the interaction among concurrently executing transactions in order to maintain the consistency of the database [41]. Implementation of concurrency control protocols in RTDBSs is difficult due to the conflicting requirements of meeting deadlines and maintaining data consistency. The RTDBS researchers, in general, assume the existence of two distinct application environments (one characterized by hard deadline transactions and the other processing soft or firm deadline transactions), and target one of the environments in their study. We therefore review the current work on concurrency control in two separate parts. In the first part, we summarize the work performed considering an application environment in which the data consistency requirement is modified since the deadline requirement cannot be relaxed. In such environments, getting timely but partially incorrect information can be preferable to getting correct but late information [57]. In the second part of this section, we examine the concurrency control approaches intended for environments where maintaining data consistency is more crucial than satisfying deadlines. Schedulers should not violate the data consistency requirement while observing the timing constraints of transactions.

### *Concurrency Control with Hard Deadline Transactions*

With the current database technology it is extremely difficult to satisfy "hard" timing constraints of transactions processed in an RTDBS. This difficulty comes from the unpredictability of transaction response times. Each transaction operation accessing a data item takes a variable amount of time due to concurrency control and disk IO [65]. In this section, we review some methods that can be used to handle the consistency and timeliness issues together in processing hard deadline transactions in RTDBSs.

Serializability is a widely accepted correctness criterion for concurrency control in database systems. Serializable schedules provide correct results and leave the database consistent. However, serializability is not a suitable technique to implement in scheduling hard deadline transactions because of the limitation of concurrency allowed by serializable executions. Existing concurrency control protocols ensuring serializability are based on either one of two techniques: blocking transactions and restarting transactions. Both techniques are inappropriate for time-critical scheduling. Blocking can cause priority inversion; i.e., a high-priority transaction (e.g., with an urgent deadline) can be blocked by a lower-priority transaction [54]. Aborting and then restarting a transaction, on the other hand, causes a waste of processing time and other system resources already used by that transaction.

The consistency model presented in [45, 71] is an attempt at the relaxation of strict serializability rules. The model is an extension of the *imprecise computation model*[4] to transaction processing in RTDBSs. In the proposed model, timing constraints are satisfied by sacrificing database consistency temporarily to some degree. *External* data consistency is defined in contrast to *internal* data consistency as maintained by conventional database systems. The external consistency constraint requires that the data used by a transaction reflect the physical environment at the time; this is in contrast to internal consistency, which requires that all data must meet some predefined constraints in the database. The model is based on the assumption that for most RTDBS applications, a timely and externally consistent result is more desirable than an out-of-date though internally consistent response. For instance, the trace of an unidentified object detected by an on-board system is externally consistent but may not be internally consistent before it is interpreted and filtered by the system.

The study of Hou et al. [30, 48, 49] also involves development of a transaction processing model to facilitate timely executions that satisfy strict deadlines. They provide a query evaluation methodology that uses statistical and heuristic time control strategies to process queries within fixed deadlines. Different degrees of accuracy (approximation) of the responses to the queries can be achieved using that methodology.

All the methods discussed so far can provide timely executions while maintaining data consistency to some extent. An extreme approach that can be acceptable in some application environments is to completely eliminate consistency checks while processing transactions. As suggested by Singhal [57], in applications where it is more important to get partially incorrect information quickly than to wait for correct information, a possible approach may be not to exercise any concurrency control and periodically examine the database for inconsistencies and restore it to a consistent state.

### *Concurrency Control with Soft/Firm Deadline Transactions*

A substantial amount of research in RTDBSs has been devoted to development of concurrency control protocols that meet soft/firm deadline requirements of transactions. The general approach taken in that research has been extending traditional concurrency-control techniques (that provide a serialization order among conflicting transactions) by applying time-critical scheduling methods to observe timing constraints of

[4]In the imprecise computation model, if a transaction does not have enough time to complete its execution, it is allowed to produce imprecise (i.e., incomplete) results from its operations [46].

transactions. A number of lock-based, optimistic and timestamp-ordering concurrency control protocols have been proposed so far. All those protocols aim to minimize the number of transactions that miss their deadlines.

Two main approaches to the lock-based real-time concurrency control have been the *Priority Inheritance* (PI) and the *Priority Abort* (PA). They are both time-cognizant extensions of the conventional two-phase locking (2PL) protocol. Variations of these approaches have been the basis for the other lock-based concurrency-control protocols.

PI was proposed by Sha et al. [54, 55] to overcome the problem of priority inversion. This scheme ensures that when a transaction blocks higher-priority transactions, it is executed at the highest priority of the blocked transactions; in other words, it inherits the highest priority. Due to the inherited priority, the transaction can be executed faster resulting in reduced blocking times for high-priority transactions.

PA prevents priority inversion by aborting low-priority transactions whenever necessary [1]. In resolving a data lock conflict, if the transaction requesting the lock has higher priority than the transaction that holds the lock, the latter transaction is aborted and the lock is granted to the former one. Otherwise, the lock-requesting transaction is blocked by the higher-priority lock-holding transaction. A high-priority transaction never waits for a lower-priority transaction. This condition prevents deadlocks if we assume that the real-time priority of a transaction does not change during its lifetime and that no two transactions have the same priority.

The performances of these two approaches have been studied by some researchers either using simulation (e.g., [2, 4, 28, 68]) or on an RTDBS testbed (e.g., [33, 34]). Although in all those works both schemes were found to perform better (i.e., satisfy more deadlines) than the conventional 2PL protocol, the results obtained for the comparative performances of the schemes do not completely agree. It was observed by Huang et al. [33, 34], Haritsa et al. [28], and Ulusoy and Belford [68] that the performance provided by PI cannot reach the level achieved by PA. Remember that PA never blocks high-priority transactions, but instead aborts low-priority transactions when necessary. It also eliminates the possibility and cost of deadlocks. The authors thus conclude that aborting a low-priority transaction is preferable in RTDBSs to blocking a high-priority one, even though aborts lead to a waste of resources. The results presented by Abbott and Garcia-Molina [2, 4], on the other hand, indicate that no protocol is the best under all conditions; the comparative performance of the schemes depends on some other factors they considered, such as the type of load, and the priority policy. Under continuous and steady load, the performance of PI was observed to be better than that of PA. This result is different from what the other researchers obtained in their experiments.

The difference is probably due to the different assumptions made and different execution models used in evaluations.

Huang et al. [33] developed a combined priority abort and priority inheritance protocol, called *conditional priority inheritance*, to capitalize on the advantages of both schemes. The protocol attempts to reduce the blocking times with respect to PI, and to reduce the abort rate with respect to PA. When a transaction $T$ is blocked by a lower priority transaction $T'$, if $T'$ is near completion, it inherits the priority of $T$; otherwise, $T'$ is aborted. The protocol assumes that the length of a transaction (i.e., the number of data items accessed by the transaction) is known in advance. The protocol has a threshold parameter $h$. At the time of a data conflict, if the remaining number of data items to be accessed by the lock-holding transaction is less than or equal to threshold $h$, then PI is applied; otherwise, PA is used. The experiments run by the authors show that the conditional priority inheritance protocol performs well for a wide range of system workloads.

An extension to PI is the *priority ceiling* protocol, proposed by Sha et al. [54, 55], which bounds the blocking time of high-priority transactions to no more than one transaction execution time. It eliminates the deadlock problem from PI and attempts to reduce the blocking delays of high-priority transactions. The "priority ceiling" of a data item is defined as the priority of the highest-priority transaction that may have a lock on that item. In order to obtain a lock on a data item, the protocol requires that a transaction $T$ must have a priority strictly higher than the highest-priority ceiling of data items locked by the transactions other than $T$. Otherwise, transaction $T$ is blocked by the transaction that holds the lock on the data item of the highest-priority ceiling. The performance of the protocol was examined in [56] using simulation. The results obtained revealed that the protocol performs poorly when the database is not memory resident. However, a significant improvement was observed in the performance when intention IO was used to prefetch data items accessed by transactions. A variant of the priority ceiling protocol, proposed by Chen and Lin [13], enables the scheduler to dynamically determine the priority ceiling of each data item. Son and Chang [59] investigated methods to apply the priority ceiling protocol as a basis for real-time locking protocol in a distributed environment.

In a more recent work, Ulusoy [66] provided a new concurrency control protocol, called *data-priority-based locking protocol*, to prove that the real-time performance provided by PA, which appears to be a good locking protocol, can be further improved if the data access requirements of transactions are known in advance. Similar to the priority ceiling protocol, the proposed protocol is based on prioritizing data items; each data item carries a priority equal to the highest priority of all transactions currently in the system that include the data item in their access lists. In order to obtain a lock on a data item $D$, the priority of a transaction $T$ must be equal to the priority of $D$. Otherwise (if the priority of $T$ is less than that of $D$), transaction $T$ is blocked by the transaction that is responsible for the priority of $D$. Suppose that $T$ has the same priority as $D$, but $D$ has already been locked by a lower-priority transaction $T'$ before $T$ arrives at the system and adjusts the priority of $D$. $T'$ is aborted at the time $T$ needs to lock $D$. Some of the transaction aborts and the resulting resource waste experienced in PA can be prevented by employing data-priority-based locking protocol. Consider the following scenario: suppose that two transactions $T_x$ and $T_y$ have conflicting accesses on item $D$, and transaction $T_x$ has higher priority. Under the new protocol, if $T_y$ tries to lock data item $D$ before $T_x$ does, the lock request of $T_y$ is not accepted. Under protocol PA, $T_y$ would get the lock on $D$, but would be aborted when the higher-priority transaction $T_x$ requests $D$. As a result, the processing time spent by $T_y$ would be simply wasted. This wasted time might have been used to help another transaction meet its deadline. Expectations about the performance of data-priority-based locking protocol were confirmed by experimental results [66, 68].

Examples of other lock-based concurrency control protocols developed for RTDBSs include those provided by Agrawal et al. [5] and by Son et al. [61]. The protocols presented in [5] were motivated by the observation that the blocking behavior of locking protocols can greatly degrade the performance of RTDBSs. A new relationship between locks, called *ordered sharing*, is used in the protocols to eliminate blocking of read and write operations at the expense of a possible delay at transaction commitment. This delay is exploited by allowing other transactions to run within the slacks of delayed transactions. In order to commit, a delayed transaction that reaches its deadline may have to abort a lower-priority transaction that has not yet completed. The protocols aim to improve the overall system performance by exploiting any available slack in a transaction. It was shown through simulation that the proposed protocols can perform better than the priority abort protocol PA.

A rather complex locking protocol was introduced by Son et al. [61] to be used in RTDBSs. In this protocol, the serialization order of active transactions is adjusted dynamically, making it possible for high-priority transactions to be executed before lower-priority transactions, while lower-priority transactions may not have to be aborted in resolving data conflicts. The problem of concurrency control is decomposed to two subproblems, namely read–write synchronization and write–write synchronization, and read–write conflicts are resolved by 2PL while write–write conflicts are resolved by the *Thomas write rule*.[5] The authors provided the results of simulation experiments examining the performance of the protocol under a wide range of workloads and data access patterns.

Some variants of the optimistic concurrency control protocol [42] have also been developed and evaluated for RTDBSs. Haritsa et al. [25] studied the relative performance of lock-based and optimistic concurrency techniques in the context of an RTDBS. PA was used as the representative lock-based protocol to be compared against the *broadcast commit* variant of the optimistic protocol. In the broadcast commit protocol, the validation check for a committing transaction is performed against the other active transactions and the transactions that are in conflict with the committing transaction are aborted. Although this protocol does not make use of transaction priorities in resolving data conflicts, it was shown to outperform the priority-based locking protocol PA over a wide range of system utilization. The observation was that transaction blocking in lock-based protocols results in unpredictable delays causing transactions to miss their deadlines. The authors later developed an optimistic protocol, called *WAIT-50*, which allows for the use of priorities to improve decision making in resolving conflicts [26]. The protocol uses a "50 percent" rule as follows: If half or more of the transactions conflicting with a committing transaction are of higher priority, the transaction is made to wait for the high-priority transactions to complete; otherwise, it is allowed to commit while the conflicting transactions are aborted. While the transaction is waiting, it is possible that it will be restarted due to the commit of one of the conflicting transactions with higher priority. WAIT-50 protocol was shown to provide significant performance gains over the broadcast commit protocol.

Huang et al. [32] implemented and evaluated a set of optimistic protocols on an RTDBS testbed. The optimistic scheme was found to perform better than the locking scheme only under low data contention. When data contention was high, the situation was reversed due to the overhead of large number of transaction restarts. Those experimental results do not agree with the simulation results of Haritsa et al. [25, 26]. The differences are contributed to the different types of systems involved in evaluations and the different degree of protocol implementation [32].

The priority inversion problem that was defined for locking protocols can also exist in an RTDBS that maintains data consistency through use of a timestamp-ordering concurrency control protocol. It is possible that a

[5]The Thomas write rule ignores a write request that has arrived late, rather than rejecting it [7].

high-priority transaction $T$ is aborted at its access to a data item, since a lower priority transaction $T'$ carrying a timestamp higher than the timestamp of $T$ has accessed that data item previously. Ulusoy [66, 68] proposed a time-cognizant concurrency control protocol that attempts to control the priority inversion problem of the timestamp-ordering scheme. In the basic timestamp-ordering protocol, scheduling decisions for conflicting operations are all based on the timestamp values assigned to transactions at startup time [7]. Each transaction is assigned a timestamp based on its submission (or resubmission) time to the system. One possible way to make use of priorities of transactions during scheduling is to involve the priorities in the timestamp assignment procedure. The new protocol categorizes the transactions into timestamp groups based on their arrival times. The time is divided into intervals of a certain length and the transactions that arrive at the system within the same interval are placed in the same timestamp group. The basic idea is to schedule the transactions of the same timestamp group based on their real-time priorities. Each transaction is assigned a two-level timestamp made up of a group timestamp and a real-time timestamp. The transactions within the same timestamp group are assigned the same group timestamp, which is the arrival time of the first transaction in that group. Real-time timestamps of transactions within the same group are determined based on the real-time priorities of transactions. The transaction with the highest priority obtains the largest real-time timestamp, so it cannot be aborted by any other transaction in the same group in the case of a data access conflict. Real-time timestamps are used in ordering the access requests of the transactions from the same group, while the group timestamp is used in ordering the transactions from different groups. It was observed through simulation experiments that the proposed protocol improves the real-time performance of the basic timestamp-ordering protocol especially under high load and high data conflict conditions; however, the improvement is not enough to bring its performance up to that of the priority-based locking protocols.

A hybrid protocol that is a combination of optimistic concurrency control and timestamp-ordering was proposed by Son et al. [62]. The protocol uses optimistic concurrency control with broadcast commit to take the advantage of the early detection and resolution of nonserializable executions. The protocol also employs dynamic timestamp allocation [6] (i.e., a transaction gradually builds its serialization order whenever a data conflict occurs) and timestamp intervals [10] (i.e., each transaction is assigned a timestamp interval instead of a single timestamp value, and the timestamp interval of each transaction is adjusted each time the transaction performs a read or a write operation). Transaction priorities are considered in deciding which transaction should be aborted if a data conflict is detected during the validation of a transaction. No results regarding the performance of the protocol have been provided by the authors.

A recent work by Hong et al. [29] introduced a *cost conscious* approach to concurrency control in RTDBSs. The cost conscious approach includes the cost of aborted transactions in priority calculation, so that the effects of transaction abort and restart overhead are considered in resolving conflicts among transactions. The dynamic priority assignment protocol provided adapts to changes in the system load to reduce the number of transaction restarts. Using simulation, it was shown that the performance of a concurrency control protocol that involves restarts in scheduling decisions can be improved by using the cost conscious approach.

O'Neil et al. [47] proposed a two-phase scheduling approach to provide more predictable transaction executions in RTDBSs. They introduced two algorithms both having two phases of transaction execution. In the first algorithm, called the *optimistic algorithm*, after performing all data access operations in the first phase, a validation step determines whether the serializability is ensured by the execution of those operations. If so, the transaction commits; otherwise, the second phase of transaction execution starts. In the second algorithm, called the *skeleton execution algorithm*, the first phase performs only calculations needed to evaluate variables used to determine the data items accessed. In the second phase of both algorithms, a real-time scheduling method is employed for transaction execution.

Performance impact of maintaining multiple versions[6] of data in RT-DBSs was studied by Kim and Srivastava [39]. They proposed several multiple-version concurrency control protocols that aim to reduce data contention and thus to increase the degree of concurrency in an RTDBS environment. The protocols are all based on the multiple-version 2PL protocol [7]. The authors claim that maintaining multiple versions may not add much to the cost of execution, because the versions may be used anyway by the recovery algorithm. The experimental results obtained using a detailed simulation model show that the protocols can provide an improvement over the single-version concurrency control protocols developed for RTDBSs.

DiPippo and Wolfe [19] developed a semantic concurrency control technique on a real-time object-oriented database system model. The semantic concurrency control technique is capable of supporting logical

[6]In a system with multiple versions of data, each write operation on a data item produces a new version rather than overwriting it.

consistency, temporal consistency, and the trade-offs between them. The technique utilizes the user-defined compatibility function of an object to determine the trade-off and to define correctness for that object. In order to invoke a method on an object, a transaction requests a semantic lock. In processing the semantic lock request, the compatibility function of the object and a set of conditions are evaluated.

## 3. DISTRIBUTED REAL-TIME DATABASE SYSTEMS

Distributed databases fit more naturally in the decentralized structures of many RTDBS applications that are inherently distributed (e.g, the stock market, banking, command and control systems, and airline reservation systems). Distributed database systems provide shared data access capabilities to transactions; i.e., a transaction is allowed to access data items stored at remote sites. While scheduling transactions in a distributed RTDBS, besides observing the timing constraints, the global consistency of the distributed database should also be preserved, as well as the local consistency at each data site [70]. To achieve this goal, it is the exchange of messages that carry scheduling information between the data sites where the transaction is being executed is required. The communication delay introduced by message exchanges constitutes a substantial overhead for the response time of a distributed transaction. Thus, guaranteeing the response times of transactions (i.e., satisfying the timing constraints), is more difficult in a distributed RTDBS than that in a single-site RTDBS. This section provides a brief overview of the recent work that has addressed various aspects of distributed RTDBSs.

### *3.1. REPLICATION*

In a *replicated database system*, copies of data can be stored redundantly at multiple sites. The potential of data replication for high data availability and improved read performance is crucial to RTDBSs. On the other hand, data replication introduces its own problems. Access to a data item is no longer controlled exclusively by a single site; instead the access control is distributed across the sites, each storing a copy of the data item. It is necessary to ensure that *mutual consistency* of the replicated data is provided; in other words, replicated copies must behave like a single copy. This can be made possible by preventing conflicting accesses on the different copies of the same data item, and by making sure that all data sites eventually receive all updates [21]. Multiple-copy updates lead to a considerable overhead due to the communication required among the data sites holding the copies.

The impact of storing multiple copies of data on satisfying timing constraints of RTDBS transactions was investigated by Ulusoy [69]. A detailed performance model of a distributed RTDBS is employed in that work to evaluate the effects of various workload parameters and design alternatives on system performance. The primary performance issue considered is the satisfaction of transaction deadlines; more specifically, an answer to the following question is sought: "does replication of data always aid in satisfying timing constraints of transactions?" Various experiments are conducted to identify the conditions under which data replication can help real-time transactions satisfy their timing constraints. Different application types are considered in evaluating the effects of the degree of data replication. Reach application is distinguished by the type (query versus update) and data access distribution (local versus remote) of the processed transactions. It was observed that replication is not attractive for update-oriented real-time applications due to the overhead of synchronizing updates on multiple copy data items. On the other hand, unless the majority of the transactions are of the update-type or the system load is high, it seems preferable to store multiple copies (but not too many) of data. The effects of site failures were also examined to estimate how much replication is needed to provide a reliable processing environment for real-time transactions of different applications.

Son and Kouloumbis [63] proposed a new replication control algorithm for distributed RTDBSs. The algorithm integrates real-time scheduling with data replication control. It employs epsilon serializability as the correctness criterion to provide more concurrency to real-time transactions. Real-time scheduling features are involved in responding to timing requirements of transactions. A token-based synchronization scheme is used to control replication. The performance issues of the algorithm were not addressed by the authors.

Lin and Lin [44] proposed some techniques to enhance the availability of replicated real-time databases. They suggest that a transaction characterized with a strict deadline should be able to execute even if the most up-to-date data copies are not available, so that the mutual consistency requirement can be relaxed for distributed RTDBSs that process hard deadline transactions. They also introduced the *user quorum* scheme to increase the availability in a partitioned RTDBS. The scheme is different from traditional quorum protocols in that it gives access rights to a partition with a majority of users rather than a partition with a majority of data copies.

### 3.2. *DISTRIBUTED CONCURRENCY CONTROL*

In a distributed database system, a scheduler at each site is responsible for controlling concurrent accesses to data items stored at that site. Access requests of both local and remote transactions are ordered together on the basis of the concurrency control protocol being executed. Distributed versions of the time-cognizant concurrency control protocols (see Section 2.4) need to be executed in distributed RTDBS environments.

The performance of distributed lock-based concurrency control protocols was studied by Ulusoy both in nonreplicated [67] and replicated [69] RTDBS environments. The distributed version of the priority-abort protocol PA was observed to perform better than the distributed priority inheritance protocol PI under various conditions in a nonreplicated RTDBS [67]. However, the difference between the performance results of protocols is not as large as that observed in a single-site RTDBS [68]. The results obtained with a replicated RTDBS, on the other hand, show that PA can beat PI only under query-based application environments and when the level of data replication is low [69]. These two different sets of results lead to the conclusion that restart-based protocols (like PA) are superior to blocking-based protocols (like PI) as long as the overhead of transaction aborts is not high. As the data become more distributed and replicated, the increased overhead of transaction aborts causes PA to perform worse than PI.

### 3.3. *SUBTASK DEADLINE ASSIGNMENT*

Kao and Garcia-Molina [37, 38] addressed the issue of the subtask deadline assignment in a distributed environment. A typical global transaction processed in a distributed system possesses subtasks (i.e., subtransactions) to be executed on various system sites. A single value of an end-to-end global deadline might not truly reflect the urgency of each individual subtask. The subtask deadlines should be earlier than the end-to-end global deadline so as to speed up the progress of the global transaction. Kao and Garcia-Molina suggested and evaluated heuristic scheduling policies for the subtask deadline assignment problem. The problem was reduced to two subproblems: one deals with *serial* subtasks (where a global transaction consists of a number of serially executing subtasks), and the other one with *parallel* subtasks (where a global transaction involves parallel execution of subtasks at different nodes).

The serial subtask problem was studied in [37]. Several ways of breaking up an end-to-end deadline into intermediate virtual deadlines that can better reflect the urgency of each subtask were discussed. One of the proposed schemes, called *equal flexibility*, tries to estimate the total amount of slack time a global transaction has and divides this slack among the subtasks proportional to their execution times. Each subtask thus has the same slack-to-execution-time ratio (flexibility). Although this method requires an estimate on execution times, it was shown that this estimation does not have to be very accurate.

The parallel subtask problem and the combined effect of serial and parallel subtask problems were studied in [38]. When a global transaction is divided into a number of subtasks for parallel processing, it is very likely that one or more subtasks run into a busy component and become tardy. This will cause the whole global transaction to miss its deadline. One scheduling heuristic proposed is based on the observation that the more subtasks of a global transaction has, the poorer is its chance of meeting its deadline. The amount of time that the transaction is allowed to finish is divided by a value that is proportional to the number of the transaction's subtasks. The larger the number of subtasks is, the earlier are the virtual deadlines assigned to the subtasks. The heuristic was shown to be quite effective for the parallel subtask problem. Combining the deadline assignment strategies proposed for both the serial and parallel subtasks, it was shown that the real-time behavior of distributed transactions can be significantly improved.

## *3.4. COMMITMENT*

The effects of a distributed transaction on the data must be visible at all sites in *all or nothing* fashion. The so-called *atomic commitment* property can be provided by a commit protocol that coordinates the subtransactions such that either all of them or none of them commit. In conventional distributed database systems, the standard approach to ensuring the atomicity property of distributed transactions is to use the *two-phase commit* (2PC) protocol [7]. It is suggested by Soparkar et al. that the unpredictability and the cost of 2PC protocol makes it unsuitable for RTDBSs. Their work in [64] is basically an investigation of possible methods to make a commit protocol *adaptive* in the sense that under different loading conditions the system can dynamically change to a different commitment strategy. In case of time delays or transient overloads, commitment protocols that relax the atomicity property can be adopted by local sites. The authors also suggest use of the concept of *compensation* for recovering from the failures of RTDBS transactions (i.e., if a transaction commits erroneously, a compensating transaction is used to perform a semantic undo). It is stated that compensation is attractive for RTDBSs because a compensation process may be deferred to be executed during periods of light system load, while traditional undo operations need to be performed immediately.

## 4. ACTIVE REAL-TIME DATABASE SYSTEMS

Conventional database systems are in general *passive*; i.e., transactions are executed only when they are explicitly initiated by a user or an application program. However, some application areas, such as automated manufacturing, air traffic control, and battle management require the underlying database system to be *active*. An *active database system* is characterized by *conditions* defined on the states of the database that need to be evaluated when predefined *events* occur, and specified *actions* that must be performed once the conditions hold [20].

If the application supported by an active database system requires timely response to critical situations, the specified actions must be executed subject to some timing constraints. Involvement of timing constraints in active databases was considered in the HiPAC (High Performance ACtive Database System) project [12, 16]. Three basic concepts explored in this project are active database management, timing constraints, and contingency plans. *Contingency plans* are defined as 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 primitives. The execution model introduces a generalized transaction model that provides correct execution of specified actions and user transactions together in a timely manner.

Korth et al. [40] introduced a new approach to the modeling of an active RTDBS. In this approach, timing constraints are associated with the states of the database rather than directly with transactions. A set of consistency constraints are also defined for the database. If a change in the database state violates a consistency constraint, a transaction is triggered to restore the consistency within a specified deadline. The deadline is determined by the timing constraint defined on that state.

## 5. ARCHITECTURAL CONSIDERATIONS

### 5.1. *INTEGRATION WITH OPERATING SYSTEMS*

Some of the basic functions performed by a database management system are also performed by an operating system. As Graham suggests

[22], real-time applications cannot afford wasteful duplication of these functions. An integration of basic building blocks of operating systems and RTDBSs is thus necessary. The functions of a RTDBS that should be supported by the underlying operating system include priority-based CPU and IO scheduling, concurrency control and recovery, buffer management, and data management.

Buchmann et al. [9] proposed a system architecture for the integration of functions from operating systems and RTDBSs. The architecture provides an interface between the operating system's scheduler and the RTDBS's concurrency control module. The priority scheduling function provided by the operating system is used through that interface to resolve data conflicts among concurrent transactions. Also, for blocking/reactivation of transactions during conflict resolution, the suspend/resume primitives of the operating system are used.

### 5.2. *MAIN MEMORY DATABASE SYSTEMS*

As pointed out before, among the most important factors that might cause a transaction to miss its deadline is the disk IO delay. One possible design approach to eliminate disk access delay from the database access is to maintain the database in main memory. Main memory databases are expected to be economically feasible in the near future due to falling memory prices and growing memory sizes [57]. The work performed so far on various design issues of conventional main memory database systems can also be adopted in RTDBSs. The research in conventional main memory databases has primarily focused on crash recovery (e.g., [23, 24]), data access methods (e.g., [18]), and query processing (e.g., [43]).

## 6. SUMMARY AND DIRECTIONS FOR FUTURE RESEARCH

The area of real-time database systems (RTDBSs) has emerged as a result of the demand to apply database technology to the management of data belonging to a real-time system. Since the amount of data handled by real-time systems has steadily been increasing, it has become essential to use efficient database management techniques for timely execution of retrieval and update operations on data. Transactions processed in an RTDBS are characterized by timing constraints, typically in the form of deadlines. Essential to RTDBSs is the processing of transactions within their deadlines while maintaining the logical consistency of data accessed by the transactions. Ideas from both real-time scheduling and database management techniques have been combined to satisfy the needs of RTDBSs.

Issues related to transaction scheduling in RTDBSs have been addressed by a number of researchers. The major contribution of the research conducted in this area has been the development of new time-cognizant protocols for concurrency control, resource scheduling, commit processing, and buffer management. These protocols have extended traditional database management techniques with time-critical scheduling methods. Although the proposed protocols have usually been tested using simulation, we believe that more experimental work is needed to demonstrate the usefulness and practicality of the protocols.

Considering the time-critical scheduling requirements of RTDBS, main memory database systems seems to be a good candidate to replace conventional database systems as they eliminate disk access delays from database access. Main memory databases can be considered as a feasible design approach to RTDBSs due to the recent advances in hardware technology that makes main memory drastically cheaper each year. However, main memory databases introduce some problems and design issues of their own [57]. As Graham suggests [22], further research is required to understand the tradeoffs of maintaining main memory databases.

We think that although a considerable amount of research has been conducted so far in the area of RTDBSs, still much work remains to be done in order to make them viable. In addition to the future research directions discussed above, other open problems in RTDBSs include the following:

- Providing a formal framework for defining a data and transaction model for RTDBSs.
- Providing language constructs to express timing constraints and to specify exception handling procedures for unsatisfied timing constraints.
- Developing performance models and benchmarks to exercise time-cognizant protocols developed for RTDBS functions.
- Efficient integration of RTDBS functions with the facilities provided by operating systems.

*I would like to thank Prof. Patrick O'Neil for his helpful comments on earlier versions of this paper.*

## REFERENCES

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

2. R. Abbott and H. Garcia-Molina, Scheduling real-time transactions with disk resident data, in *Proceedings of the 15th International Conference on Very Large Data Bases*, 1989, pp. 385–396.
3. R. Abbott and H. Garcia-Molina, Scheduling I/O requests with deadlines: A performance evaluation, in *Proceedings of the 11th Real-Time Systems Symposium*, 1990, pp. 113–124.
4. R. Abbott and H. Garcia-Molina, Scheduling real-time transactions: A performance evaluation, *ACM Trans. Database Systems* 17(3):513–560 (1992).
5. D. Agrawal, A. El Abbadi, and R. Jeffers, Using delayed commitment in locking protocols for real-time databases, in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1992, pp. 104–113.
6. R. Bayer, K. Elhardt, J. Heigert, and A. Reiser, Dynamic timestamp allocation for transactions in database systems, in *Proceedings of the 2nd International Symposium on Distributed Databases*, 1982, pp. 9–20.
7. P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, MA, 1987.
8. S. R. Biyabani, J. A. Stankovic, and K. Ramamritham, The integration of deadline and criticalness in hard real-time scheduling, in *Proceedings of the 9th Real-Time Systems Symposium*, 1988, pp. 152–160.
9. A. P. Buchmann, D. R. McCarthy, M. Shu, and U. Dayal, Time-critical database scheduling: A framework for integrating real-time scheduling and concurrency control, in *Proceedings of the 5th International Conference on Data Engineering*, 1989, pp. 470–480.
10. C. Boksenbaum, M. Cart, J. Ferrie, and J. Pons, Concurrent certifications by intervals of timestamps in distributed database systems, *IEEE Trans. Software Eng.* SE-13(4):409–419 (1987).
11. M. J. Carey, R. Jauhari, and M. Livny, Priority in DBMS resource scheduling, in *Proceedings of the 15th International Conference on Very Large Data Bases*, 1989, pp. 397–410.
12. S. Chakravarthy et al., HiPAC: A research project in active, time-constrained database management, Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, 1989.
13. M. Chen and K. J. Lin, Dynamic priority ceilings: A concurrency control protocol for real-time systems, *Real-Time Systems* 2(4):325–346 (1990).
14. S. Chen, J. A. Stankovic, J. Kurose, and D. Towsley, Performance evaluation of two new disk scheduling algorithms for real-time systems, *Real-Time Systems* 3(3):307–336 (1991).
15. H. T. Chou and D. DeWitt, An evaluation of buffer management strategies for relational database systems, in *Proceedings of the 11th International Conference on Very Large Data Bases*, 1985, pp. 127–141.
16. U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ledin, D. McCarthy, A. Rosenthal, S. Sarin, M. J. Carey, M. Livny, and R. Jauhari, The HiPAC project: Combining active database and timing constraints, *ACM SIGMOD Record* 17(1):51–70 (1988).
17. H. M. Deitel, *An Introduction to Operating Systems*, Addison-Wesley, Reading, MA, 1984.
18. D. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebreaker, and D. Wood, Implementation techniques for main memory database systems, in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1984, pp. 1–78.

19. L. C. DiPippo and V. F. Wolfe, Objected-based semantic real-time concurrency control, in *Proceedings of the 14th IEEE Real-Time Systems Symposium*, 1993.
20. R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 2nd ed., Benjamin/Cummings, Redwood City, CA, 1994.
21. H. Garcia-Molina and R. K. Abbott, Reliable distributed database management, in *Proc. IEEE* 75(5):601–620 (1987).
22. M. H. Graham, Issues in real-time data management, *Real-Time Systems* 4(3):185–202 (1992).
23. L. Gruenwald and M. H. Eich, MMDB reload algorithms, in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1991, pp. 397–406.
24. R. Hagmann, A crash recovery scheme for a memory resident database system, *IEEE Trans. Computers* C-35(9):839–843 (1986).
25. J. R. Haritsa, M. J. Carey, and M. Livny, On being optimistic about real-time constraints, in *Proceedings of the ACM SIGACT-SIGMOD-SIGART*, 1990, pp. 331–343.
26. J. R. Haritsa, M. J. Carey, and M. Livny, Dynamic real-time optimistic concurrency control, in *Proceedings of the 11th Real-Time Systems Symposium*, 1990, pp. 94–103.
27. J. R. Haritsa, M. J. Carey, and M. Livny, Value-based scheduling in real-time database systems, Technical Report 1204, Department of Computer Science, University of Wisconsin-Madison, 1991.
28. J. R. Haritsa, M. J. Carey, and M. Livny, Data access scheduling in firm real-time database systems, *Real-Time Systems* 4(3):203–241 (1992).
29. D. Hong, T. Johnson, and S. Chakravarthy, Real-time transaction scheduling: A cost conscious approach, in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1993, pp. 197–206.
30. W. C. Hou, G. Özsoyoğlu, and B. K. Taneja, Processing aggregate queries with hard time constraints, in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1989, pp. 68–8.
31. J. Huang, J. A. Stankovic, D. Towsley, and K. Ramamritham, Experimental evaluation of real-time transaction processing, in *Proceedings of the 10th Real-Time Systems Symposium*, 1989, pp. 144–153.
32. J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley, Experimental evaluation of real-time optimistic concurrency control schemes, in *Proceedings of the 17th International Conference on Very Large Data Bases*, 1991, pp. 35–46.
33. J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley, On using priority inheritance in real-time databases, in *Proceedings of the 12th Real-Time Systems Symposium*, 1991, pp. 210–221.
34. J. Huang, J. A. Stankovic, K. Ramamritham, D. Towsley, and B. Purimetla, Priority inheritance in soft real-time databases, *Real-Time Systems* 4(3):243–268 (1992).
35. R. Jauhari, M. J. Carey, and M. Livny, Priority-hints: An algorithm for priority-based buffer management, in *Proceedings of the 16th International Conference on Very Large Data Bases*, 1990, pp. 708–721.
36. E. D. Jensen, C. D. Locke, and H. Tokuda, A time-driven scheduling model for real-time operating systems, in *Proceedings of the 6th Real-Time Systems Symposium*, 1985, pp. 112–122.
37. B. Kao and H. Garcia-Molina, Deadline assignment in a distributed soft real-time system, in *Proceedings of the 13th International Conference on Distributed Computing Systems*, 1993, pp. 428–437.

38. B. Kao and H. Garcia-Molina, Subtask deadline assignment for complex distributed soft real-time tasks, Technical Report STAN-CS-93-1491, Department of Computer Science, Stanford University, 1993.
39. W. Kim and J. Srivastava, Enhancing real-time DBMS performance with multiversion data and priority based disk scheduling, in *Proceedings of the 12th Real-Time Systems Symposium*, 1991, pp. 222–231.
40. H. F. Korth, N. Soparkar, and A. Silberschatz, Triggered real-time databases with consistency constraints, in *Proceedings of the 16th International Conference on Very Large Data Bases*, 1990, pp. 71–82.
41. H. F. Korth and A. Silberschatz, *Database System Concepts*, 2nd ed., Computer Science Series, McGraw-Hill, New York, 1991.
42. H. T. Kung and J. T. Robinson, On optimistic methods for concurrency control, *ACM Trans. Database Systems* 6(2):213–226 (1981).
43. T. Lehman and M. J. Carey, Query processing in main memory database management systems, in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1986, pp. 239–250.
44. K. J. Lin and M. J. Lin, Enhancing availability in distributed real-time databases, *ACM SIGMOD Record* 17(1):34–43 (1988).
45. K. J. Lin, Consistency issues in real-time database systems, in *Proceedings of the 22nd Hawaii International Conference on Systems Sciences*, 1989, pp. 654–661.
46. J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao, Algorithms for scheduling imprecise ccmputations, *IEEE Computer* 24(5):58–68 (1991).
47. P. E. O'Neil, K. Ramamritham, and C. Pu, A two-phase approach to predictably scheduling real-time transactions, to appear in *Performance of Concurrency Control Algorithms in Centralized Database Systems*, V. Kumar (ed.), Prentice-Hall, Englewood Cliffs, NJ, 1994.
48. G. Özsoyoğlu, Z. M. Özsoyoğlu, and W. C. Hou, Research in time and error-constrained database query processing, in *Proceedings of the 7th IEEE Workshop on Real-Time Operating Systems and Software*, 1990, pp. 32.
49. G. Özsoyoğlu, K. Du, S. Guruswamy, and W. C. Hou, Processing real-time, non-aggregate queries with time-constraints in CASE-DB, in *Proceedings of the 8th International Conference on Data Engineering*, 1992, pp. 410–417.
50. H. Pang, M. Livny, and M. J. Carey, Transaction scheduling in multiclass real-time database systems, Technical Report 1110, Department of Computer Science, University of Wisconsin-Madison, 1992.
51. H. Pang, M. J. Carey, and M. Livny, Managing memory for real-time queries, in *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, 1994, pp. 221–232.
52. K. Ramamritham, Real-time databases, *Distributed Parallel Databases* 1(2) (1993).
53. C. Ruemmler and J. Wilkes, An introduction to disk drive modeling, *IEEE Computer* 27(3):17–28 (1994).
54. L. Sha, R. Rajkumar, and J. Lehoczky, Concurrency control for distributed real-time databases, *ACM SIGMOD Record* 17(1):82–98 (1988).
55. L. Sha, R. Rajkumar, and J. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, *IEEE Trans. Computers* 39(9):1175–1185 (1990).
56. L. Sha, R. Rajkumar, S. H. Son, and C. H. Chang, A real-time locking protocol, *IEEE Trans. Computers* 40(7):793–800 (1991).
57. M. Singhal, Issues and approaches to design of real-time database systems, *ACM SIGMOD Record* 17(1):19–33 (1988).

58. S. H. Son (ed.), Special Issue on Real-Time Databases, *ACM SIGMOD Record* (1988).
59. S. H. Son and C. H. Chang, Performance evaluation of real-time locking protocols using a distributed software prototyping environment, in *Proceedings of the 10th International Conference on Distributed Computing Systems*, 1990, pp. 124–131.
60. S. H. Son, Scheduling real-time transactions, in *Proceedings of the EUROMICRO Workshop on Real-Time Systems*, 1990, pp. 25–32.
61. S. H. Son, S. Park, and Y. Lin, An integrated real-time locking protocol, in *Proceedings of the 8th International Conference on Data Engineering*, 1992, pp. 527–534.
62. S. H. Son, J. Lee, and Y. Lin, Hybrid protocols using dynamic adjustment of serialization order for real-time concurrency control, *Real-Time Systems* 4(3):269–276 (1992).
63. S. H. Son and S. Kouloumbis, Replication control for distributed real-time database systems, in *Proceedings of the 12th International Conference on Distributed Computing Systems*, 1992, pp. 144–151.
64. N. Soparkar, E. Levy, H. F. Korth, and A. Silberschatz, Adaptive commitment for real-time distributed transactions, Technical Report TR-92-15, Department of Computer Science, University of Texas at Austin, 1992.
65. J. A. Stankovic and W. Zhao, On real-time transactions, *ACM SIGMOD Record* 17(1):4–18 (1988).
66. Ö. Ulusoy, Concurrency control in real-time database systems, Technical Report UIUCDCS-R-92-1762, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.
67. Ö. Ulusoy and G. G. Belford, Real-time lock based concurrency control in a distributed database system, in *Proceedings of the 12th International Conference on Distributed Computing Systems*, 1992, pp. 136–143.
68. Ö. Ulusoy and G. G. Belford, Real-time transaction scheduling in database systems, *Information Systems* 18(8):559–580 (1993).
69. Ö. Ulusoy, Processing real-time transactions in a replicated database system, *Distributed Parallel Databases* 2(4):405–436 (1994).
70. Ö. Ulusoy, A study of two transaction processing architectures for distributed real-time database systems, To appear in *Journal of Systems Software* (1995); also Technical Report, BU-CEIS-94-22, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey.
71. S. V. Vrbsky and K. J. Lin, Recovering imprecise transactions with real-time constraints, in *Proceedings of the 7th Symposium on Reliable Distributed Systems*, 1988, pp. 185–193.

*Received 3 October 1994; revised 13 February 1995.*