Real-Time Databases and Data Services

Krithi Ramamritham, Indian Institute of Technology Bombay, krithi@cse.iitb.ac.in
Sang H. Son, University of Virginia, son@cs.virginia.edu
Lisa Cingiser DiPippo, University of Rhode Island, dipippo@cs.uri.edu

1 Introduction

Typically, a real–time system consists of a *a controlling system* and a *controlled system*. In an automated factory, the controlled system is the factory floor with its robots, assembling stations, and the assembled parts, while the controlling system is the computer and human interfaces that manage and coordinate the activities on the factory floor. Thus, the controlled system can be viewed as the *environment* with which the computer interacts.

The controlling system interacts with its environment based on the data available about the environment, say from various sensors, e.g. temperature and pressure sensors. It is imperative that the state of the environment, as perceived by the controlling system, be consistent with the actual state of the environment. Otherwise, the effects of the controlling systems' activities may be disastrous. Hence, timely monitoring of the environment as well as timely processing of the sensed information is necessary. The sensed data is processed further to derive new data. For example, the temperature and pressure information pertaining to a reaction may be used to derive the rate at which the reaction appears to be progressing. This derivation typically would depend on past temperature and pressure trends and so some of the needed information may have to be fetched from archival storage. Based on the derived data, where the derivation may involve multiple steps, actuator commands are set. For instance, in our example, the derived reaction rate is used to determine the amount of chemicals or coolant to be added to the reaction. In general, the history of (interactions with) the environment are also logged in archival storage.

In addition to the timing constraints that arise from the need to continuously track the environment, timing correctness requirements in a real–time (database) system also arise because of the need to make data available to the controlling system for its decision-making activities. If the computer controlling a robot does not command it to stop or turn on time, the robot might collide with another object on the factory floor. Needless to say, such a mishap can result in a major catastrophe.

Besides robotics, applications such as medical patient monitoring, programmed stock trading, and military command and control systems like submarine contact tracking require timely actions as well as the ability to access and store complex data that reflects the state of the application's environment. That is, data in these applications must be valid, or $fresh$, when it is accessed in order for the application to perform correctly. In a patient monitoring system, data such as heart rate, temperature, and blood pressure must be collected periodically. Transactions that monitor the danger level of a patient's status must be performed within a specified time, and the data must be accessed within an interval that defines the validity of the data. If not, the computations made by the transactions do not reflect the current state of the patient's health.

A traditional database provides some of the functionality required by these applications, such as coordination of concurrent actions and consistent access to shared data. But they do not provide for enforcement of the

timing constraints imposed by the applications. A Real-Time DataBase (RTDB) has all of the features of a more traditional database, but it can also express and maintain time-constrained data and time-constrained transactions. In the last fifteen years, there has been a great deal of research towards developing real-time databases that provide this functionality. In more recent years, much of the research that has been performed in the area of RTDBs has been applied, and extended in related fields, generally known as real-time data services. This paper surveys the highlights of the research that has been performed to advance the state-of-the art of RTDBs and real-time data services.

In developing RTDB solutions that provide the required timeliness of data and transactions, there are various issues that must be considered. Below is a discussion of some of the concerns that have been the subject of research in this field.

Data, transaction and system characteristics. A RTDB must maintain not only the logical consistency of the data and transactions, it must also satisfy transaction timing properties as well as data temporal consistency. Transaction timing constraints include deadlines, earliest start times and latest start times. Transactions must be scheduled such that these constraints are met. Data temporal consistency imposes constraints on how old a data item can be and still be considered valid. There are various ways to characterize real-time transactions. If an application requires that the timing constraints on transactions be met in order to be correct, these transactions are considered to be hard. If a firm transaction misses a timing constraint, it no longer has value to the system, and therefore can be aborted. A soft transaction provides some, likely reduced, value to the system after its constraints have been violated. Transactions can also be characterized by their timing requirements. That is, a transaction can be periodic, or aperiodic. Finally, a RTDB system can be static or dynamic. In a static system, all data and requirements are known a priori, and the transactions can be designed and analyzed for schedulability up front. A dynamic system does not have this a priori knowledge, and therefore must be able to adapt to changing system requirements.

Scheduling and transaction processing. Scheduling and transaction processing techniques that consider data and transaction characteristics have been a major part of the research that has been performed in the field of RTDBs. Many of them consider the inherent tradeoffs between quality of data vs. timeliness of processing. For example, in a weather monitoring RTDB, it may be necessary to sacrifice image processing time and store lower quality satellite images of areas that have no important weather activity going on. This will allow for more important areas, such as those with hurricane activity, to be more precise. Further, this will also allow for transactions accessing these important images to meet their access timing constraints. For another example, in a programmed stock trading application, a transaction that updates a stock price may be blocked by another transaction that is reading the same piece of data, and has a shorter deadline. If the stock price in question is getting old it would be in the interest of its temporal consistency to allow the updating transaction to execute. However, this execution could violate the logical consistency of the data or of the reading transaction. Thus, there is a trade-off between maintaining temporal consistency and maintaining logical consistency. If logical consistency is chosen, then there is the possibility that a stock price may become old, or that a transaction may miss a deadline.

If, on the other hand, temporal consistency is chosen, the consistency of the data or of the transactions involved may be compromised.

Distribution. With the advent of high-speed, relatively low-cost networking, many of the applications that require a RTDB are not located on a single computer. Rather, they are distributed, and may require that the real-time data be distributed as well. Issues involved with distributing data include data replication, replication consistency, distributed transaction processing, and distributed concurrency control. When real-time requirements are added, these systems become much more complex. For instance, in a collaborative combat planning application where sensor data is collected by several ships in a region, and various decision-makers are viewing the collected data, it is critical that all collaborators share the same view of the scenario. The degree of distribution is also an issue that researchers have been examining recently. That is, a distributed RTDB that exists on three stationary nodes will have different requirements than a RTDB made up of hundreds of mobile computers each of which can sense local data and share it with the others.

Quality of Service and Quality of Data. It is often not possible to maintain the logical consistency of a database and the temporal consistency as well. Therefore, there must be a trade-off made to decide which is more important. In dynamic systems, where it is not possible to know and control the real-time performance of the database, it may be necessary to accept levels of service that are lower than optimal. It may also be necessary to trade off the quality of the data in order to meet specified timing constraints. The RTDBs must provide mechanisms to specify allowable levels of service and quality of data, and it must also provide a way to adjust these levels when it is necessary.

This paper describes the important research results that have been produced in the above areas. It starts out in Section 2 by defining data freshness and timing properties. Section 3 presents research that has been done in the area of transaction processing for RTDBs. As mentioned above, this is where much of the RTDB focus has been in the past. Quality of service (QoS), and also quality of data (QoD) issues are addressed in Section 4. In Section 5, we discuss some new applications for which RTDB research has been useful. Finally, Section 6 concludes by discussing challenges that still exist for this research area. It also proposes a research agenda for those interested in doing continuing work in this field.

2 Data Freshness and Timing Properties

The need to maintain consistency between the actual state of the environment and the state as reflected by the contents of the database leads to the notion of *temporal consistency*: the need to maintain coherency between the state of the environment and its reflection in the database. This arises from the need to keep the controlling system's view of the state of the environment consistent with the actual state of the environment.

A real-time database (RTDB) is composed of real-time objects which are updated by periodic sensor transactions. An *object* in the database models a real world entity, for example, the position of an aircraft. A real-time object is one whose state may become invalid with the passage of time. Associated with the state is a temporal validity interval. To monitor the states of objects faithfully, a real-time object must be refreshed by a sensor transaction

before it becomes invalid, i.e., before its temporal validity interval expires. The actual length of the temporal validity interval of a real-time object is application dependent.

Here, we do not restrict the notion of sensor data to the data provided by physical sensors. Instead, we consider a broad meaning of sensor data. Any data item, whose value reflects the time-varying real-world status, is considered a sensor data item.

Sensor transactions are generated by intelligent sensors which periodically sample the value of real-time objects. When sensor transactions arrive at RTDBs with sampled data values, their updates are installed and real-time data are refreshed. So one of the important design goals of RTDBs is to guarantee that temporal data remain fresh, i.e., they are always valid. RTDBs that do not keep track of temporal validity of data may not be able to react to abnormal situations in time. Therefore, efficient design approaches are desired to guarantee the freshness of temporal data in RTDBs while minimizing the CPU workload resulting from periodic sensor transactions.

Whereas in the rest of this section we will be dealing with sensor transactions, in particular, with the derivation of their timing properties, it must be pointed out that many RTDBs also possess *user transactions*, transactions that access temporal as well as nontemporal data. User transactions typically arrive aperiodically. User transactions do not write any temporal data object, but they can read/write non-temporal data and their execution time and data access pattern can be time-varying. For example, in a decision support system a transaction can read different sets of dynamic, i.e., temporally varying, data and perform different operations according to the situation.

From here on, $\mathcal{T} = \{\tau_i\}_{i=1}^{m}$ refers to a set of periodic sensor transactions $\{\tau_1, \tau_2, .., \tau_m\}$ and $\mathcal{X} = \{X_i\}_{i=1}^{m}$ refers to a set of temporal data. All temporal data are assumed to be kept in main memory. Associated with X_i $(1 \leq i \leq m)$ is a validity interval of length α_i: transaction τ_i $(1 \leq i \leq m)$ updates the corresponding data X_i. Because each sensor transaction updates different data, no concurrency control is considered for sensor transactions. We assume that a sensor always samples the value of a temporal data at the beginning of its period, and all the first instances of sensor transactions are initiated at the same time. C_i, D_i and P_i $(1 \leq i \leq m)$ denote the execution time, relative deadline, and period of transaction τ_i, respectively. D_i of sensor transaction τ_i is defined as follows:

$$D_i = \text{Deadline of } \tau_i \text{ - Sampling Time of } \tau_i.$$

Deadlines of sensor transactions are firm deadlines. The goal of a RTDB designer is to determine P_i and D_i such that all the sensor transactions are schedulable and CPU workload resulting from sensor transactions is minimized. Let us assume a simple execution semantics for periodic transactions: a transaction must be executed once every period. However, there is no guarantee on when an instance of a periodic transaction is actually executed within a period. We also assume that these periodic transactions are preemptable.

Assume the period and relative deadline of a sensor transaction are set to be equal to the data validity interval. Because the separation of the execution of two consecutive instances of a transaction can exceed the validity interval, data can become invalid under this *One-One* approach. So this approach cannot guarantee the freshness of temporal data in RTDBs.

Example 2.1: Consider Figure 1: A periodic sensor transaction τ_i with deterministic execution time C_i refreshes

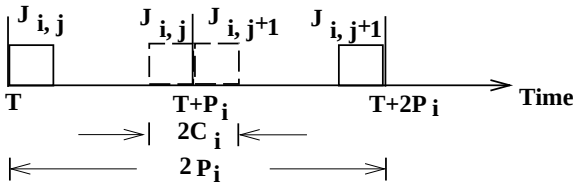

Figure 1. extreme execution cases of periodic sensor transactions

temporal data X_i with validity interval α_i. The period P_i and relative deadline D_i of τ_i are assigned the value α_i. Suppose $J_{i,j}$ and $J_{i,j+1}$ are two consecutive instances of sensor transaction τ_i. Transaction instance $J_{i,j}$ samples data X_i with validity interval $[T, T+\alpha_i)$ at time T, and $J_{i,j+1}$ samples data X_i with validity interval $[T+\alpha_i, T+2\alpha_i)$ at time $T+\alpha_i$. From Figure 1, the actual arrival time of $J_{i,j}$ and finishing time of $J_{i,j+1}$ can be as close as $2C_i$, and as far as $2P_i$, i.e., $2\alpha_i$ when the period of τ_i is α_i. In the latter case, the validity of data X_i refreshed by $J_{i,j}$ expires after time $T+\alpha_i$. Since $J_{i,j+1}$ cannot refresh data X_i before time $T+\alpha_i$, X_i is invalid from $T+\alpha_i$ until it is refreshed by $J_{i,j+1}$, just before the next deadline $T+2\alpha_i$.

In order to guarantee the freshness of temporal data in RTDBs, the period and relative deadline of a sensor transaction are each typically set to be less than or equal to one-half of the data validity interval [95, 60]. In Figure 1, the farthest distance (based on the arrival time of a periodic transaction instance and the finishing time of its next instance) of two consecutive sensor transactions is $2P_i$. If $2P_i \leq \alpha_i$, then the freshness of temporal data X_i is guaranteed as long as instances of sensor transaction τ_i do not miss their deadlines.

Unfortunately, even though data freshness is guaranteed, this design approach at least doubles CPU workload of the sensor transaction in the RTDBs compared to the *One-One* approach. Next, we introduce the *More-Less* approach whose goal is to minimize the CPU workload of sensor transactions while guaranteeing the freshness of temporal data in RTDBs. Recall that, for simplicity of discussion, we have assumed that a sensor transaction is responsible for updating a single temporal data item in the system. In *More-Less*, the period of a sensor transaction is assigned to be *more* than *half* of the validity interval of the temporal data updated by the transaction, while its corresponding relative deadline is assigned to be *less* than *half* of the validity interval of the same data. However, the sum of the period and relative deadline always equals the length of the validity interval of the data updated. Consider Figure 2. Let $P_i > \frac{\alpha_i}{2}$, $C_i \leq D_i < P_i$ where $P_i + D_i = \alpha_i$. The farthest distance (based on the arrival time of a periodic transaction instance and the finishing time of its next instance) of two consecutive sensor transactions J_{ij} and J_{ij+1} is $P_i + D_i$. In this case, the freshness of X_i can always be maintained if sensor transactions make their deadlines. Obviously, the load incurred by sensor transaction τ_i can be reduced if P_i is enlarged (which implies that D_i is shrunk.). Therefore, we have the constraints $C_i \leq D_i < P_i$ and $P_i + D_i = \alpha_i$ which aim at minimizing the CPU workload of periodic transaction τ_i.

Example 2.2: Suppose there is temporal data X_i with validity interval α_i in a uniprocessor RTDB system. τ_i updates X_i periodically. For simplicity of discussion, we assume that $\delta_i = 0$. Our goal is to assign proper values to P_i and D_i given C_i and α_i so as to reduce the CPU workload resulting from sensor transaction τ_i. Suppose

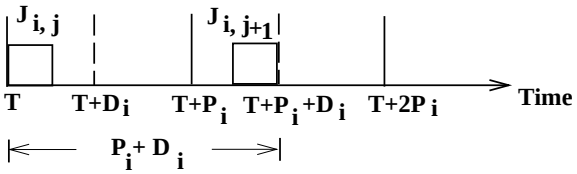

Figure 2. Illustration of *More-Less* approach

Approach	D_i	P_i	**Workload**
One-One	α_i	α_i	$U_o = \frac{C_i}{\alpha_i} = \frac{1}{5}$
Half-Half	$\frac{\alpha_i}{2}$	$\frac{\alpha_i}{2}$	$U_h = \frac{2C_i}{\alpha_i} = \frac{2}{5}$
More-Less	$\frac{\alpha_i}{3}$	$\frac{2\alpha_i}{3}$	$U_m = \frac{3C_i}{2\alpha_i} = \frac{3}{10}$

Table 1. Comparison of three approaches

$C_i = \frac{1}{5}\alpha_i$, possible values of P_i, D_i and the corresponding CPU workload according to the three different design approaches are shown in Table 1.

Only *Half-Half* and *More-Less* can guarantee the freshness of temporal data X_i if all the sensor transactions complete before their deadlines. We also notice that $U_o < U_m < U_h$ (see Table 1). If $P_i = \frac{N-1}{N}\alpha_i$, then $D_i = \frac{1}{N}\alpha_i$, where $N \geq 2$. The freshness of temporal data in RTDBs is guaranteed if all sensor transactions complete before their deadlines. In such a case, we also notice that $U_m = \frac{NC_i}{(N-1)\alpha_i}$ and $U_o \leq U_m < U_h$. Theoretically, if $N \to \infty$, $U_m \to U_o$.

Unfortunately, how close U_m can get to U_o depends on C_i since $D_i \geq C_i$ implies $\frac{\alpha_i}{C_i} \geq N$. As N increases, relative deadlines become shorter and sensor transactions are executed with more stringent time constraints.

Therefore, given a set of sensor transactions in RTDBs, we need to find periods and deadlines of update transactions based on the temporal validity intervals of data such that the CPU workload of sensor transactions is minimized and the schedulability of the resulting sensor transactions is guaranteed. The *More-Less* approach achieves this, as shown in [143]. Recent work [60] on similarity-based load adjustment also adopts this approach to adjust periods of sensor transactions based on similarity bounds.

Concepts from active databases can be used to maintain the temporal validity of derived objects, where a derived object's value is dependent on the values of associated sensor data and it is updated sporadically. The technique described in [135] is one such approach and is designed to ensure that maintaining temporal consistency does not result in a significant deterioration of the real-time performance with respect to meeting transaction deadlines.

Before we close this section, we would like to point out that effective scheduling of real-time transactions is greatly hampered by the large variance that typically exists between average-case and worst-case execution times in a database system. This unpredictability arises due to a variety of factors including data-value dependent execution patterns, blocking or restarts due to data contention, and disk latencies that are a function of the access

history. [97] attempts to eliminate some of these sources of unpredictability and thereby facilitate better estimation of transaction running times. It employs main-memory database technology and presents new logical and physical versioning techniques that ensure that read-only transactions are completely isolated from update transactions with respect to data contention.

3 Transaction processing

In this section, issues related to the timely processing of transactions and queries such that they produce logically correct database states as well as outputs. Section 3.1 deals with scheduling and concurrency control issues, Section 3.2 summarizes results in the commit processing area, Section 3.3 presents work on recovery, and Section 3.4 outlines approaches developed to deal with resources such as I/O and buffers.

3.1 Scheduling and Concurrency Control

In this section, we discuss various aspects of transaction and query processing where the transactions and queries have time constraints attached to them and there are different consequences of not satisfying those constraints.

A key issue in transaction processing is *predictability*. In the context of an individual transaction, this relates to the question: "will the transaction meet its time-constraint"? Section 3.1.1 deals with the processing of transactions that have hard deadlines,, i.e., deadlines that must be met, and 3.1.2 deals with transactions that have soft deadlines, i.e., deadlines which can be missed, but the larger the misses, the lower the value accrued to the system.

3.1.1 Dealing with Hard Deadlines

All transactions with hard deadlines must meet their time constraints. Since dynamically managed transactions cannot provide such a guarantee, the data and processing resources as well as time needed by such transactions have to be guaranteed to be made available when necessary. There are several implications.

Firstly, we have to know when the transactions are likely to be invoked. This information is readily available for periodic transactions, but for aperiodic transactions, by definition, it is not. The smallest separation time between two incarnations of an aperiodic transaction can be viewed as its period. Thus, we can cast all hard real-time transactions as *periodic* transactions.

Secondly, in order to ensure a priori that their deadlines will be met, we have to determine their resource requirements and worst-case transaction execution times. This requires that many restrictions be placed on the structure and characteristics of real-time transactions.

Once we have achieved the above, we can treat the transactions in a manner similar to the way real-time systems treat periodic tasks that require guarantees, i.e., by using static *table-driven* schedulers or preemptive *priority-driven* approaches. Static *table-driven* schedulers reserve specific time slots for each transaction. If a transaction does not use all of the time reserved for it, the time may be reclaimed to start other hard real-time transactions earlier than planned. Otherwise, it can be used for soft real-time transactions or left idle. The table-driven approach

is obviously very inflexible. One priority-driven approach is the rate-monotonic priority assignment policy. One can apply the schedulability analysis tools associated with it to check if a set of transactions are schedulable given their periods and data requirements. This is the approach discussed in [104] where periodic transactions that access main memory resident data via read and write locks are scheduled using rate-monotonic priority assignment.

We mentioned earlier that the variance between the worst-case computational needs and actual needs must not be very large. We can see why. Since the schedulability analysis is done with respect to worst-case needs, if the variance is large, many transactions that may be doable in the average case will be considered infeasible in the worst-case. Also, if the table-driven approach is used, a large variance will lead to large idle times.

In summary, while it is possible to deal with hard real-time transactions using approaches similar to those used in real-time systems, many restrictions have to be placed on these transactions so that their characteristics are known a priori. Even if one is willing to deal with these restrictions, poor resource utilization may result given the worst-case assumptions made about the activities.

3.1.2 Dealing with Soft Transactions

With soft real-time transactions, we have more leeway to process transactions since we are not required to meet the deadlines all the time. Of course, the larger the number of transactions that meet their deadlines the better. When transactions have different values, the value of transactions that finish by their deadlines should be maximized. The complexity involved in processing real-time transactions comes from these goals. That is to say, we cannot simply let a transaction run, as we would in a traditional database system, and abort it should its deadline expire before it commits. We must meet transaction deadlines by adopting priority-assignment policies and conflict resolution mechanisms that explicitly take time into account. Note that priority assignment governs CPU scheduling and conflict resolution determines which of the many transactions contending for a data item will obtain access. As we will see, conflict resolution protocols make use of transaction priorities and because of this, the priority assignment policy plays a crucial role. We also discuss the performance implications of different deadline semantics.

Scheduling and Conflict Resolution Rather than assigning priorities based on whether the transactions are CPU or I/O (or data) bound, real-time database systems must assign priorities based on transaction time constraints and the value they impart to the system. Possible policies are:

- *Earliest-deadline-first.*
- *Highest-value-first.*
- *Highest-value-per-unit-computation-time-first.*
- *Longest-executed-transaction-first*

It has been shown that the priority assignment policy has significant impact on performance and that when different transactions have different values, both deadline *and* value must be considered [45].

For the purpose of conflict resolution in real-time database systems, various *time-cognizant* extensions of two phase locking, optimistic, and timestamp based protocols have been proposed in the literature. See for example, [1, 18, 37, 45, 47, 46, 71, 114, 124]. Some of these are discussed below.

In the context of two-phase locking, when a transaction requests a lock that is currently held by another transaction we must take into account the characteristics of the transactions involved in the conflict. Considerations involved in conflict resolution are the deadline and value (in general, the priority) of transactions, how long the transactions have executed, and how close they are to completion. Consider the following set of protocols:

- If a transaction with a higher priority is forced to wait for a lower priority transaction to release the lock, a situation known as *priority inversion* arises. This is because a lower priority transaction makes a higher priority transaction to wait. In one approach to resolving this problem, the lock holder *inherits* the lock requester's priority whereby it completes execution sooner than with its own priority.
- If the lock holding transaction has lower priority, abort it. Otherwise let the lock requester wait.
- If the lock holding transaction is closer to its deadline, lock requester waits, independent of its priority.

Priority Inheritance is shown to reduce transaction blocking times [47]. This is because the lock holder executes at a higher priority (than that of the waiting transaction) and hence finishes early, thereby blocking the waiting higher priority transaction for a shorter duration. However, even with this policy, the higher priority transaction is blocked, in the worst case, for the duration of a transaction under strict two phase locking. As a result, the priority inheritance protocol typically performs even worse than a protocol that makes a lock requester wait independent of its priority.

If a higher priority transaction always aborts a low priority transaction, the resulting performance is sensitive to data contention. On the other hand, if a lower priority transaction that is closer to completion inherits priority rather than aborting, then a better performance results even when data contention is high. Such a protocol is a combination of the abort-based protocol proposed for traditional databases and the priority-inheritance protocol proposed for real-time systems [105]. Said differently, the superior performance of this protocol [47] shows that even though techniques that work in real-time systems on the one hand and database systems on the other hand may not be applicable directly, they can often be tailored and adapted to suit the needs of real-time database systems. It should be noted that abort-based protocols (as opposed to wait-based) are especially appropriate for real-time database systems because of the time constraints associated with transactions.

Let us now consider optimistic protocols. In protocols that perform backward validation, the validating transaction either commits or aborts depending on whether it has conflicts with transactions that have already committed. The disadvantage of backward validation is that it does not allow us to take transaction characteristics into account. This disadvantage does not apply to forward validation. In forward validation, a committing transaction usually aborts ongoing transactions in case they conflict with the validating transaction. However, depending on the characteristics of the validating transaction and those with which it conflicts, we may prefer not to commit the validating transaction. Several policies have been studied in the literature [37, 38, 46]. In one, termed *wait-50*, a

validating transaction is made to wait as long as more than half the transactions that conflict with it have earlier deadlines. This is shown to have superior performance.

Time-cognizant extensions to timestamp-based protocols have also been proposed. In these, when data accesses are out of timestamp order, the conflicts are resolved based on their priorities. In addition, several combinations of locking-based, optimistic and timestamp-based protocols have been proposed but require quantitative evaluation [71].

Exploiting multiple versions of data for enhanced performance has been addressed in [55]. Multiple versions can reduce conflicts over data. However, if data must have temporal validity, old versions which are outdated must be discarded. Also, when choosing versions of related data, their relative consistency requirements must be taken into account: consider a transaction that uses multi-versioned data to display aircraft positions on an air-traffic controller's screen. The data displayed must have both absolute validity as well as relative validity.

Different transaction semantics are possible with respect to discarding a transaction once its deadline is past. For example, with firm deadlines, a late transaction is aborted once its deadline expires [38]. In general, with soft deadlines, once a transaction's value drops to zero, it is aborted [45]. On the other hand, in the transaction model assumed in [1], all transactions have to complete execution even if their deadlines have expired. In this model, delayed transactions may cause other transactions also to miss their deadlines and this can have a cascading effect.

Experimental studies reported in the literature cited at the end of this article are very comprehensive and cover most aspects of real-time transaction processing, but most have not considered time constraints associated with data.

Database systems in which time validity intervals are associated with the data are discussed in [121, 58, 59]. Such systems introduce the need to maintain data temporal consistency in addition to logical consistency. The performance of several concurrency control algorithms for maintaining temporal consistency are studied in [121]. In the model introduced in [121], a real-time system consists of periodic tasks which are either read-only, write-only or update (read-write) transactions. Data objects are temporally inconsistent when their ages or dispersions are greater than the absolute or relative thresholds allowed by the application. Studies of two-phase locking and optimistic concurrency control algorithms, as well as rate-monotonic and earliest deadline first scheduling algorithms show that the performances of the rate-monotonic and earliest deadline first algorithms are close when the load is low. At higher loads, earliest deadline first outperforms rate-monotonic when maintaining temporal consistency. They also observed that optimistic concurrency control is generally worse at maintaining temporal consistency of data than lock based concurrency control, even though the former allows more transactions to meet their deadlines. It is pointed out in [121] that it is difficult to maintain the data and transaction time constraints due to the following reasons:

- A transient overload may cause transactions to miss their deadlines.
- Data values may become out of date due to delayed updates.
- Priority based scheduling can cause preemptions which may cause the data read by the transactions to

become temporally inconsistent by the time they are used.

- Traditional concurrency control ensures logical data consistency, but may cause temporal data inconsistency.

Motivated by these problems, and taking into account the fact that transactions process data with validity constraints and such data will be refreshed with sensor transactions, the notion of *data-deadline* is introduced in [142]. Informally, data-deadline is a deadline assigned to a transaction due to the temporal constraints of the data accessed by the transaction. Further, a *forced wait* policy which improves performance significantly by forcing a transaction to delay further execution until a new version of sensor data becomes available is also proposed.

In [58], a class of real-time data access protocols called SSP (Similarity Stack Protocols) is proposed. The correctness of SSP is based on the concept of similarity which allows different but sufficiently timely data to be used in a computation without adversely affecting the outcome. SSP schedules are deadlock free, subject to limited blocking and do not use locks. In [59], weaker consistency requirements based on the similarity notion are proposed to provide more flexibility in concurrency control for data-intensive real-time applications. While the notion of data similarity is exploited in their study to relax serializability (hence increase concurrency), in [142], it is coupled with data-deadline and used to improve the performance of transaction scheduling. The notion of similarity is used to adjust transaction workload by Ho et. al. [60], and incorporated into embedded applications (e.g., process control) in [22].

3.2 Distributed Databases and Commit Protocols

Many real-time database applications are inherently distributed in nature. These include the intelligent network services database, telecom databases, mobile telecommunication systems and the 1-800 telephone service in the United States. More recent applications include the directory, data feed and electronic commerce services that have become available on the World Wide Web. The performance, reliability, and availability of such applications can be significantly enhanced through the replication of data on multiple sites of the distributed network.

An algorithm for maintaining consistency and improving the performance of replicated DRTDBS is proposed in [115]. In this algorithm, a multiversion technique is used to increase the degree of concurrency. Replication control algorithms that integrate real-time scheduling and replication control are proposed in [116]. In contrast to the relaxed correctness assumed by the latter, conventional one-copy serializability supported by the algorithm called MIRROR reported in [141]. MIRROR (Managing Isolation in Replicated Real-time Object Repositories), is a concurrency control protocol specifically designed for firm-deadline applications operating on replicated real-time databases. MIRROR augments the classical O2PL concurrency control protocol with a state-based realtime conflict resolution mechanism. In this scheme, the choice of conflict resolution method is a dynamic function of the states of the distributed transactions involved in the conflict. A feature of the design is that acquiring the state knowledge does not require inter-site communication or synchronization, nor does it require modifications to the two-phase commit protocol.

The performance of the classical distributed 2PL locking protocol (augmented with the priority abort (PA) and

priority inheritance (PI) conflict resolution mechanisms) and of OCC algorithms in replicated DRTDBS was studied in the literature for real-time applications with soft deadlines. The results indicate that 2PL-PA outperforms 2PL-PI only when the update transaction ratio and the level of data replication are both low. Similarly, the performance of OCC is good only under light transaction loads. Making clear-cut recommendations on the performance of protocols in the soft deadline environment is rendered difficult, however, by the following: (1) There are two metrics: Missed Deadlines and Mean Tardiness, and protocols which improve one metric usually degrade the other, (2) The choice of the post-deadline value function has considerable impact on relative protocol performance, and (3) There is no inherent load control, so the system could enter an unstable state.

Temporal consistency guarantees are also studied in distributed real-time systems. In [147], *Distance constrained scheduling* is used to provide temporal consistency guarantees for real-time primary-backup replication service.

Let us now consider the transaction commitment process. Once a transaction reaches its commit point, it is better to let it commit quickly so that its locks can be released soon. If commit delays are not high, which will be the case in a centralized database, the committing transaction can be given a high enough priority so that it can complete quickly. The solution is not so easy in a distributed system because of the distribution of the commitment process. Furthermore, since a deadline typically refers to the deadline until the end of the two-phase commit, but since the decision on whether or not to commit is taken in the first phase, we can enter the second phase only if we know that it will complete before the deadline. This requires special handling of the commit process. An alternative is to associate the deadline with the beginning of the second phase, but this may delay subsequent transactions since locks are not released until the second phase. A few papers in the literature [122, 145] have tried to address the issue of distributed commit processing but have required either relaxing the traditional notion of atomicity or strict resource allocation and resource performance guarantees from the system.

In [41], its authors propose and evaluate a commit protocol that is specifically designed for the real-time domain without these changes. PROMPT allows transactions to optimistically borrow in a controlled manner, the updated data of transactions currently in their commit phase. This controlled borrowing reduces the data inaccessibility and the priority inversion that is inherent in distributed real-time commit processing. A simulation-based evaluation shows PROMPT to be highly successful as compared to the classical commit protocols in minimizing the number of missed transaction deadlines. In fact its performance is close to the best on-line performance that could be achieved using the optimistic lending approach.

3.3 Recovery Issues

Recovery is a complex issue even in traditional databases and is more so in real-time database systems for two reasons. Firstly, the process of recovery can interfere with the processing of ongoing transactions. Specifically, suppose we are recovering from a transaction aborted due to a deadline miss. If locks are used for concurrency control, it is important to release them as soon as possible so that waiting transactions can proceed without delay so as to meet their deadlines. However, it is also necessary to undo the changes done by the transaction to the data if in-place updates are done. But this consumes processing time that can affect the processing of transactions

that are not waiting for locks to be released. Whereas optimistic concurrency control techniques or a shadow-pages based recovery strategy can be used to minimize this time, they have several disadvantages including lack of commercial acceptance. Secondly, unlike traditional databases where permanent data should always reflect a consistent state, in real-time databases, the presence of temporal data, while providing some opportunities for quicker recovery [134], adds to the complexities of the recovery of transactions. Specifically, if a transaction's deadline expires before it completes the derivation of a data item, then rather than restoring the state of the data to its previous value, it could declare the data to be invalid thereby disallowing other transactions from using the value. The next instance of the transaction, in case the data is updated by a periodic transaction, may produce a valid state.

There is a need for designing new algorithms for logging and recovery in RTDBs because the sequential nature of logging and the lack of time and priority cognizance during recovery are not in tune with the priority oriented and preemptive nature of activities in RTDBs. Motivated by this need, [107] presents SPLIT, a partitioned logging and recovery algorithm for RTDBs, and ARUN (Algorithms for Recovery Using Non-Volatile High Speed Store), a suite of logging and recovery algorithms for RTDBs. In order to improve performance, ARUN makes use of the Solid State Disk (SSD) technology that is referred to as Non-Volatile High Speed Store (NVHSS).

The logging and recovery algorithms are based on dividing data into a set of equivalence classes derived from a taxonomy of data and transaction attributes. For example, data can be broadly classified into Hot and Cold depending on the type of the data (like critical, temporal etc.), and type of the transactions (like high priority, low priority etc.), that accesses the data. The logging and recovery algorithms assume that the classes of data are disjoint and that the recovery of one class can be done without having to recover the other classes, and if one class is recovered, then the transactions accessing that class can proceed without having to wait for the system to recover the other classes. For instance, network management systems consist of different kinds of data such as the performance, traffic, configuration, fault and security management data. The fault management data, which could potentially get updated frequently, is very critical to the operation that recovering it immediately after a crash can improve the performance of the system. A recent work towards achieving bounded and predictable recovery using real-time logging based on these ideas is presented in [110].

3.4 Managing I/O and Buffers

While the scheduling of CPU and data resources has been studied fairly extensively in the real-time database literature, studies of scheduling approaches for dealing with other resources, such as disk I/O, and buffers has begun only recently. In this section we review some recent work in this area and discuss some of the problems that remain.

I/O scheduling is an important area for real-time systems given the large difference in speeds between CPU and disks and the resultant impact of I/O devices' responsiveness on performance. Since the traditional disk scheduling algorithms attempt to minimize average I/O delays, just like traditional CPU scheduling algorithms aim to minimize average processing delays, time-cognizant I/O scheduling approaches are needed.

It must be recognized that what is important is the meeting of transaction deadlines and not the individual deadlines that may be attached to I/O requests. Assume that we model a transaction execution as a sequence of (disk I/O, computation) pairs culminating in a set of disk I/O's, the latter arising from writes to log and to the changed pages. Suppose we assign (intermediate) deadlines to the I/O requests of a transaction given the transaction's deadline. One of the interesting questions with regard to disk I/O scheduling is: How does one derive the deadline for an I/O request from that of the requesting transaction? First of all, it must be recognized that depending on how these I/O deadlines are set, deadlines associated with I/O requests may be *soft* since even if a particular I/O deadline is missed, the transaction may still complete by its deadline. This is the case if I/O deadlines are set such that the overall laxity (i.e., the difference between the time available before the deadline and the total computation time) of a transaction is uniformly divided among the computations and the I/O. On the other hand, assume that an intermediate deadline is equal to the latest completion time (i.e., the time an I/O must complete assuming that subsequent computations and I/O are executed without delay). This is the less preferred method since we now have a firm deadline associated with I/O requests – if an I/O deadline is missed, there is no way for the transaction to complete by its deadline and so the requesting transaction must be aborted.

Work on I/O scheduling includes [19, 2, 21]. The priority driven algorithm described in [19] is a variant of the traditional SCAN algorithm which works on the elevator principle to minimize disk arm movement. Without specifying how priorities are assigned to individual I/O requests, [19] proposes a variant in which the SCAN algorithm is applied to each priority level. Requests at lower priority are serviced only after those at higher priority are served. Thus, if after servicing a request, one or more higher priority requests are found waiting, the disk arm moves towards the highest priority request that is closest to the current disk arm position. In the case of requests arising from transactions with deadlines, priority assignment could be based on the deadline assigned to the I/O request.

Another variant of SCAN, one which directly takes I/O deadlines into account is FD-SCAN [2]. In this algorithm, given the current position of the disk arm, the disk arm moves towards the request with the earliest deadline that can be serviced in time. Requests that lie in that direction are serviced and after each service it is checked whether (1) a request with an even earlier deadline has arrived and (2) the deadline of the original result cannot be met. In either case, the direction of disk arm movement may change.

Clearly, both these protocols involve checks after each request is served and so incur substantial run-time overheads. The protocols described in [21] are aimed at avoiding the impact of these checks on I/O performance. Specifically, the protocols perform the necessary computations while I/O is being performed. In the SSEDO algorithm (Shortest-seek and Earliest Deadline by Ordering), the need to give higher priority to requests with earlier deadlines is met while reducing the overall seek times. The latter is accomplished by giving a high priority to requests which may have large deadlines but are very close to the current position of the disk arm. A variant of SSEDO is SSEDV which works with specific Deadline Values, rather than Deadline Orderings. [21] shows how both the algorithms can be implemented so as to perform disk scheduling while service is in progress and shows that the algorithms have better performance than the other variants of the SCAN algorithms.

Another resource for which contention can arise is the database buffer. What we have here is a conflict over buffer slots – akin to conflicts that occur over a time slot, in the case of a CPU. Thus, similar issues arise here also. Specifically, how to allocate buffer slots to transactions and which slots to replace when a need arises are some of the issues. Consider buffer replacement: in case there is a need to replace an existing buffer slot to make room for a new entry, the replacement policy may have an impact on performance, especially if the slot being replaced is used by an uncommitted transaction. Studies discussed in [19] show that transaction priorities must be considered in buffer management.

4 Satisfying QoS/QoD Requirements

In many important real-time applications including e-commerce, agile manufacturing, sensor networks, traffic control, target tracking, and network management, transactions should be processed within their deadlines, using fresh (temporally consistent) sensor data that reflect the current real-world status. Meeting both requirements of timeliness and data freshness is challenging. Generally, transaction execution time and data access pattern are not known a priori, but could vary dynamically. For example, transactions in stock trading may read varying sets of stock prices, and perform different arithmetic/logical operations to maximize the profit considering the current market status. Further, transaction timeliness and data freshness can often pose conflicting requirements. By preferring user requests to sensor updates, the deadline miss ratio is improved; however, the data freshness might be reduced. Alternatively, the freshness increases if updates receive a higher priority. In this section, we first review previous work in Quality of Service (QoS) management in RTSBs, discuss the performance metrics for QoS management, and then present several approaches based on feedback control in RTDBs.

4.1 Earlier Work

Despite the abundance of the QoS research, QoS-related work is relatively scarce in database systems. Priority Adaptation Query Resource Scheduling (PAQRS) provided timeliness differentiation of query processing in a memory-constrained environment [89]. From the observation that the performance of queries can vary significantly depending on the available memory, per-class query response time was differentiated by an appropriate memory management and scheduling. Given enough memory, queries can read the operand relations at once to produce the result immediately. If less memory is allocated, they have to use temporary files to save the intermediate results, therefore, the query processing may slow down. In this way, query deadline miss ratios were differentiated between the classes. However, no data freshness issues were considered, and the performance could easily fluctuate under the workload changes.

An on-line update scheduling policy has been proposed in the context of the web server [62]. The performance of a web server can be improved by caching dynamically generated data at the web server and the back-end database continuously updates them. Given the views to materialize, the proposed update scheduling policy can significantly improve the data freshness compared to FIFO scheduling. A complementary problem of view materialization in

web service is discussed in [61]. While the paper considers the trade-offs between response time and data freshness, it provides neither miss ratio nor data freshness guarantees.

Stanford Real-Time Information Processor (STRIP) addressed the problem of balancing between the freshness and transaction timeliness [3]. To study the trade-off between freshness and timeliness, four scheduling algorithms were introduced to schedule updates and transactions, and the performance was compared. In their later work, a similar trade-off problem was studied for derived data [4]. Ahmed et al. proposed a new approach to maintain the temporal consistency of derived data [5]. Different from STRIP, an update of a derived data object is explicitly associated with a certain timing constraint, and is triggered by the database system only if the timing constraint could be met. By a simulation study, the relative performance improvement was shown compared to the forced delay scheme of STRIP. None of the two approaches considers dynamic adaptations of update policy, and no performance guarantee is provided.

The correctness of answers to database queries can be traded off to enhance the timeliness. A query processor, called APPROXIMATE [133], can provide approximate answers depending on the availability of data or time. An imprecise computation technique (milestone approach [70]) is applied by APPROXIMATE. In the milestone approach, the accuracy of the intermediate result increases monotonically as the computation progresses. Therefore, the correctness of answers to the query could monotonically increase as the query processing progresses. A relational database system called CASE-DB [88] can produce approximate answers to queries within certain deadlines. Approximate answers are provided processing a segment of the database by sampling, and the correctness of answers can improve as more data are processed. Before beginning each data processing, CASE-DB determines if the segment processing can be finished in time. In replicated databases, consistency can be traded off for shorter response time. For example, epsilon serializability [93] allows a query processing despite the concurrent updates. Notably, the deviation of the answer to the query can be bounded, different from a similar approach called quasi serializability [26]. An adaptable security manager is proposed in [119], in which the database security level can be temporarily degraded to enhance timeliness. These performance trade-off schemes lack a systematic QoS management architecture and none of them consider providing guarantees for both miss ratio and data freshness.

4.2 QoS Metrics for RTDBs

Before we discuss QoS management and performance metrics for real-time data services, it is important to note that almost all the work on QoS in RTDBs is not aiming to provide hard guarantees. Instead, the main focus is on soft real-time applications, in which infrequent deadline misses/temporal consistency violations can be tolerated, if neither of them exceeds the limits specified in the QoS requirements.

Two major performance metrics usually considered for QoS specification are the following:

- *User Transaction Deadline Miss Ratio:* In a QoS specification, a database administrator can specify the target deadline miss ratio that can be tolerated for a specific real-time application.
- *Data Freshness:* We categorize data freshness into *database freshness* and *perceived freshness*. Database

freshness is the ratio of fresh data to the entire temporal data in a database. Perceived freshness is the ratio of of fresh data accessed to the total data accessed by timely transactions. To measure the current perceived freshness, we exclusively consider timely transactions. This is because tardy transactions, which missed their deadlines, add no value to the system. Note that only the perceived freshness can be specified in the QoS requirement. A QoS-sensitive approach provides the perceived freshness guarantee while managing the database freshness internally (hidden to the users)[51].

Long-term performance metrics such as ones listed above are not sufficient for performance specification of dynamic systems, in which the system performance can be widely time-varying. For that reason, transient performance metrics such as overshoot and settling time, adopted from control theory, have been considered as performance metrics for real-time data services [31, 92, 79]. Similar transient performance metrics are proposed in [99] to capture the responsiveness of adaptive resource allocation in real-time systems.

- *Overshoot* is the worst-case system performance in the transient system state (e.g., the highest miss ratio in the transient state). Overshoot is an important metric because a high transient miss ratio can cause serious implications in several applications such as robots and e-commerce.
- *Settling time* is the time for the transient overshoot to decay and reach the steady state performance. The settling time represents how fast the system can recover from overload. This metric has also been called reaction time or recovery time.

4.3 Feedback Control-based QoS Management

As one of the first efforts to address QoS management in RTDBs, Kang et al. [50, 51] developed a novel QoS management architecture called **QMF** (a QoS management architecture for deadline Miss ratio and data Freshness). Figure 3 shows the QMF architecture. In QMF, a formal control theoretic approach is applied to compute the required workload adjustment considering the miss ratio error, i.e., the difference between the desired miss ratio and the currently measured miss ratio. Feedback control is used since it is very effective to support the desired performance when the system model includes uncertainties [80, 92]. At each sampling instant, the feedback-based miss ratio controller measures the miss ratio and computes the miss ratio error, and the controller computes the control signal, i.e., the required workload adjustment to react to the error.

According to the required workload adjustment computed in the feedback loop, flexible freshness management and admission control are applied to support the desired miss ratio without affecting the freshness. Using the notion of perceived freshness, an *adaptive update policy* was developed to maintain the freshness in a cost-effective manner. Initially, all data are updated immediately when their new sensor readings arrive. Under overload conditions, some sensor data can be updated with reduced update rates, if necessary, to improve the miss ratio (as long as the target perceived freshness is supported). This flexible approach contrasts to the existing database update policy, commonly accepted in the real-time database research such as [3, 54], which is fixed and not adaptable regardless of the current system status.

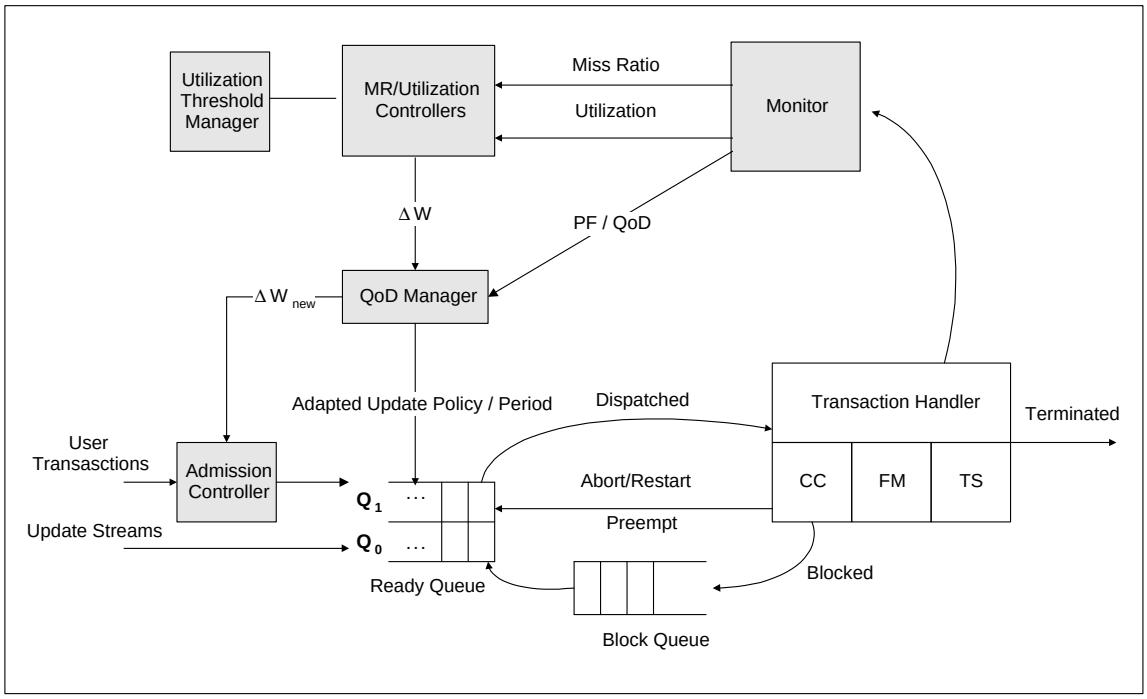

Figure 3. RTDBS architecture for QoS management using feedback control

To prevent potential deadline misses or stale data accesses due to the delay for on-demand updates, Kang et al developed an alternative approach in which all data are updated immediately [50]. In that approach, the notions of *QoD (Quality of Data)* and *flexible validity intervals* are used to manage the freshness. When overloaded, update periods of some sensor data can be relaxed within the specified range of QoD to reduce the update workload, if necessary. However, sensor data are always maintained fresh in terms of flexible validity intervals. Therefore, the age of sensor data is always bounded. According to the performance study, QMF shows satisfying stringent QoS requirements for a wide range of workloads and access patterns. QMF achieves a near zero miss ratio and perfect freshness even given dynamic workloads and data access patterns.

4.4 QoS Management using Imprecision

Imprecise computation techniques [74] provide means for achieving graceful degradation during transient overloads by trading off resource needs for the quality of a requested service. Imprecise computation and feedback control scheduling have been used for QoS management of RTDBs [6, 7, 8]. In this approach the notion of imprecise computation is applied on transactions as well as data, i.e., data objects are allowed to deviate, to a certain degree, from their corresponding values in the external environment. Although a RTDB models an external environment that changes continuously, the values of data objects that are slightly different in age or in precision are often interchangeable as consistent read data for user transactions. The time it takes to update a data object alone introduces a time delay which means that the value of the data object cannot be the same as the corresponding

real-world value at all times. This means that during transient overloads the system skips update transactions with values similar to the values stored in the database. To measure data imprecision the notion of data error, denoted de_i, is used which gives an indication of how much the value of a data object d_i stored in the RTDB deviates from the corresponding real-world value given by the latest arrived transaction updating d_i. A user transaction T_j is discarded if the data error of the data object d_i to be updated by T_j is less or equal to the maximum data error mde (i.e. $de_i \leq mde$). If mde increases, more update transactions are discarded, degrading the precision of the data. Imprecision at transaction level can be expressed in terms of certainty, accuracy, and specificity [148]. Certainty refers to the probability of a result to be correct, accuracy refers to the degree of accuracy of a value returned by an algorithm (typically through a bound on the difference from the exact solution), and specificity reflects the level of detail of the result. The imprecision of the result of a user transactions T_i, denoted the transaction error te_i, increases as the resource available for the transaction decreases.

The database operator expresses QoS requirements, in terms mde and average te_i of terminated transactions (denoted ate), by specifying not only the desired steady-state performance, but also the transient-state performance describing the worst-case system performance and system adaptability in the face of unexpected failures or load variation.

The general outline of the QoS management using imprecision is the following: Admitted transactions are placed in the ready queue. Periodically, ate is monitored and fed into the QoS controller, which compares the desired ate with the actual ate to get the current performance error. Based on this the controller computes a change, denoted δl, to the total estimated requested load. Based on δl, the quality of data (QoD) manager changes the total estimated requested load by adapting mde. The precision controller discards an update transaction writing to a data object d_i having an error less or equal to the maximum data error allowed, i.e. $de_i \leq mde$. The portion of δl not accommodated by the QoD manager, denoted δl_{new}, is returned to the admission controller (AC), which enforces the remaining load adjustment.

4.5 QoS Management for Distributed RTDBS

Providing quality-of-service guarantees for data services in a distributed environment is a challenging task. The presence of multiple sites in distributed environments raises issues that are not present in centralized systems. The transaction workloads in distributed real-time databases may not be balanced and the transaction access patterns may be time-varying and skewed. The work in [136] describes an algorithm that provides quality-of-service guarantees for data services in distributed real-time databases with full replication of temporal data. The algorithm consists of feedback-based local controllers and heuristic global load balancers (GLB) working at each site. The local controller controls the admission process of incoming transactions. The global load balancers collect the performance data from different nodes and balance the system-wide workload.

At each node, there are a local miss ratio controller and a local utilization controller. The local miss ratio controller takes the miss ratios from the latest sampling period, compares them with the QoS specification and computes the local miss ratio control signal, which is used to adjust the target utilization at the next sampling

period. In order to prevent under-utilization, a utilization feedback loop is added. It is used to avoid a trivial solution, in which all the miss ratio requirements are satisfied due to under-utilization. At each sampling period, the local utilization controller compares the utilization of the last period with the preset utilization threshold and generates the local utilization control signal. The local miss ratio controller reduces the incoming transaction workload when the QoS specification is violated; the utilization increases the incoming transaction workload when the system is under-utilized.

To balance the workload between the nodes and thus provide distributed QoS management, decentralized global load balancers (GLB) are used. *GLB*s sit at each node in the system, collaborating with each other for load balancing. At each sampling period, nodes in the system exchange their system performance data. The *GLB* at each node compares the system condition of different nodes. If a node is overloaded while some other nodes in the system are not, in the next period, the *GLB* at the overloaded node will send some transactions to the nodes that are not overloaded.

A more recent work extends the algorithm so that it scales to large distributed systems [137]. In that work, partial data replication and efficient replica management algorithms are studied because full replication is inefficient or impossible in large distributed systems.

5 New and Emerging Applications

Much of the research described in the previous sections has dealt with the application of real-time techniques and theory to databases. Recently, there has been a trend towards applying the results of this core RTDB research to other, related applications. The key ingredients of these applications are the requirements for real-time response to requests for real-time data. Sensor networks are a natural application for real-time data services because their main purpose is to provide sensed data to some requesting entity, very often with real-time constraints on the data and on the requests. A similar application that has recently benefited from early RTDB research is real-time mobile databases in which some or all of the nodes in the database can change location and are connected via a wireless network. Both of these applications bring new constraints to be considered. They both must deal with the unpredictability and lower bandwidth of wireless networks. They also both may need to consider power issues in some or all of the devices used in the system. A third application that has recently become a subject of research is real-time web-based data services. These applications include programmed stock trading and information services. The research involved in providing real-time data from such services pulls from various areas already discussed in this paper. A related area of research deals with dissemination of streaming real-time data. A key goal in these applications is delivering temporally valid real-time data to requestors within specified timing constraints. This section will discuss how each of these new application areas have provided new challenges to consider along with the RTDB issues discussed earlier. It will also describe how these applications have leveraged the existing research and developed new solutions to tackle some of these challenges.

5.1 Sensor Network Applications

Sensor networks are large-scale wireless networks that consist of numerous sensor and actuator nodes used to monitor and interact with physical environments [28][43]. From one perspective sensor networks are similar to distributed database systems. They store environmental data on distributed nodes and respond to aperiodic and long-lived periodic queries [15][48][83]. Data interest can be pre-registered to the sensor network so that the corresponding data is collected and transmitted only when needed. These specified interests are similar to views in traditional databases because they filter the data according to the application's data semantics and shield the overwhelming volume of raw data from applications [16][106].

Sensor networks have inherent real-time properties. The environment that sensor networks interact with is usually dynamic and volatile. The sensor data usually has an absolute validity interval of time after which the data values may not be consistent with the real environment. Transmitting and processing "stale" data wastes communication resources and can result in wrong decisions based on the reported out-of-date data. Besides data freshness, often the data must also be sent to the destination by a deadline. To date, not much research has been performed on real-time data services in sensor networks.

Despite their similarity to conventional distributed real-time data-bases, sensor networks differ in the following important ways. First, individual sensors are small in size and have limited computing resources, while they also must operate for long periods of time in an unattended fashion. This makes power conservation an important concern in prolonging the lifetime of the system. In current sensor networks, the major source of power consumption is communication. To reduce unnecessary data transmission from each node, data collection and transmission in sensor networks are always initiated by subscriptions or queries. Second, any individual sensor is not reliable. Sensors can be damaged or die after consuming the energy in the battery. The wireless communication medium is also unreliable. Packets can collide or be lost. Because of these issues we must build trust on a group of sensor nodes instead of any single node. Previous research emphasizes reliable transmission of important data or control packets at the lower levels, but less emphasis is on the reliability on data semantics at the higher level [98]. Third, the large amount of sensed data produced in sensor networks necessitates in-network processing. If all raw data is sent to base stations for further processing, the volume and burstiness of the traffic may cause many collisions and contribute to significant power loss. To minimize unnecessary data transmission, intermediate nodes or nearby nodes work together to filter and aggregate data before the data arrives at the destination. Fourth, sensor networks can interact with the environment by both sensing and actuating. When certain conditions are met, actuators can initiate an action on the environment. Since such actions are difficult to undo, reducing false alarms is crucial in certain applications.

There are many ongoing data service middleware research projects for the sensor network applications, including Cougar, Rutgers Dataman, SINA, SCADDS, Smart-msgs, and some virtual-machine-like designs [23][24][102][109][16] [30][84][106]. COUGER and SINA are two typical data-centric middleware designs which have goals that are similar to our design goal of providing data services. In COUGER, sensor data is viewed as tables and query execution plans are developed and possibly optimized in th middleware. DSWare project [69] is more tailored to sensor

networks, including supporting group-based decision, reliable data-centric storage, and implementing other approaches to improve the performance of real-time execution, reliability of aggregated results and reduction of communication. SINA is a cluster-based middleware design which focuses on the cooperation among sensors to conduct a task. Its extensive SQL-like primitives can be used to issue queries in sensor networks. However, it does not provide schemes to hide the faulty nature of both sensor operations and wireless communication. In SINA it is the application layer that must provide robustness and reliability for data services. The real-time scheduling component and built-in real-time features of other service components make DSWare more suitable than SINA for real-time applications in wireless sensor networks.

Multisensor data fusion research focuses on solutions that fuse data from multiple sensors to provide more accurate estimation of the environment [49][94]. In mobile-agent-based data fusion approaches, software that aggregates sensor information are packed and dispatched as *mobile agents* to "hot" areas (e.g., the area where an event occurred) and work independently there. The software migrates among sensors in a cluster, collects observations, then infers the real situation [94]. This group-based approach makes use of consensus among a number of nearby sensors of the same type to increase the reliability of a single observation. The mobile-agent-based approach, however, leverages on the migration traffic of mobile agents and their appropriate processing at each sensor node in its routes. For instance, if a node in the route inserts wrong data or refuses to forward the mobile agents, the aggregation and subsequent analysis are untrustful.

A fuzzy modelling approach is sometimes used for data fusion in sensor networks. It is used to model the uncertainty in sensor failures and faulty observations [101]. This approach is useful in modelling the sensor error rates due to equipment wear and aggregating local decisions from multiple sensors that measure the same type of data. Some optimal decision schemes focus on the fusion of asynchronously arriving decisions [20][100]. E. Bosse et. al. presented a modelling and simulation approach for a real-time algorithm in multi-source data fusion systems [17]. These data fusion schemes are suitable for increasing the accuracy of decisions, but require extensive computing resources. Dempster-Shafer evidential theory is also applied to incorporate uncertainty into decisions in some sensor fusion research [86]. This scheme uses *Belief* and *Plausibility* functions to describe the reliability feature of each source and uses a normalized Dempster's combination rule to integrate decisions from different sources. The *confidence function* in DSWare is similar to Dempster-Shafer method except that it places the evidence in both temporal and spatial spectrums to take the real-time validity intervals of data and possible contexts into consideration.

5.2 Mobile Real-Time Databases

Recent advances in wireless communications technology have made mobile information services a valuable mechanism for applications to share and distribute data. Applications like real-time traffic information systems, and mobile internet stock trading systems have requirements for timely transactions and valid data. However, the resource constraints of mobile systems make it difficult to meet timing constraints. These constraints include low bandwidth, unreliable wireless networks and frequent disconnections. The research described in this section at-

tempts to apply RTDB solutions and solve the resource constraint issues for mobile real-time database applications.

The work in [52] describes a mobile real-time database system model that consists of a number of fixed and mobile hosts. Each mobile host has a fixed host that acts as a mobile support station (MSS) and coordinates the operations of the transactions submitted by that mobile host. A mobile transaction in this model can execute by the submitting mobile host, and/or at one or more fixed hosts, and has a deadline by which it must be completed. The mobile transaction can be executed either in a single request message to the fixed network, or as a sequence of messages for Read and Write operations to the fixed network, thus placing the majority of the execution of the transaction either on the fixed host (former case) or on the mobile host (latter case). The authors performed various simulations to test these execution strategies, as well as other chosen techniques. The results of these simulations indicate that as the number of hosts increases, the fraction of transactions that satisfy their deadlines decreases. Further, transactions that are executed on the fixed host, consistently perform better than those that are executed on the mobile host.

The work in [65], extends a RTDB concurrency control technique to handle the specific characteristics of a mobile RTDB. The protocols in this work address two main issues: the cost of resolving data conflicts, and the impact of the mobile network. The system model consists of both a mobile network and a reliable wired network. The concurrency control mechanisms described in this paper are based on the HP-2PL protocol [1]. The Distributed High Priority Two Phase Locking protocol (DHP-2PL) uses a transaction restart mechanism to resolve data conflicts. However, it attempts to minimize the restarting of transactions that access data on mobile hosts by allowing a committing transaction to hold a lock until it has finished the commit procedure. Priority inversion is minimized by increasing the priority of the committing transaction higher than all executing transactions. To handle the problem of disconnection of a mobile transaction, a cautious waiting scheme is used in which a higher priority transaction is restarted by a lower priority transaction if the higher priority transaction is suspected of having been disconnected.

The idea of transaction shipping is presented in [65] to reduce communication costs of executing mobile transactions. Recall that in [52], it was shown that transactions that are executed on the fixed host perform better than those executed on the mobile host. Transaction shipping sends the entire transaction to the fixed database server. The main issue of concern is how to identify the execution path and the required data objects of a transaction ahead of time. A pre-analysis approach is suggested in which the set of operations of the transaction is identified, and the precedence relationships of the operations are determined. Given this information, the database server can execute the transaction without the need to communicate with the mobile host on which it originated.

In [66], a multi-version data model is adopted to allow for more flexibility in resolving data conflicts. This provides a looser correctness criterion based on relative consistency, and therefore increases data availability in the mobile system. In this model, new updates of data items are generated by mobile hosts, and a specific fixed host maintains a backup copy of, not only the current version, but also of previous copies of the data item. An image transaction is generated at a fixed host, called the anchor host, when a mobile transaction is received. Its job is to get its required data items to the anchor host in time for the mobile transaction to access it. Techniques

similar to those in [65] are used for pre-analysis. When the mobile transaction is executed, the relative consistency of the data items is checked to ensure that they all reflect the same temporal state of the data. After the mobile transaction accesses a version of a particular data item, all other data items are checked for relative consistency. If at any point, such a version cannot be found, the transaction is restarted, and the next latest version of its first accessed data item will be selected.

An enhanced multi-version data model is presented in [68] for data broadcasts in time constrained mobile systems. The broadcast cycle is made smaller by relaxing the requirement to broadcast every data item in every cycle. For each version of a data item in the broadcast cycle, this technique keeps track of the cycle number when the version was created, and the cycle number of the latest version of the data item. This allows for the distance between versions to be recorded. This work also proposes an on-demand broadcast for minimizing missing deadlines. After the broadcast of all data items that are scheduled, the system examines the on-demand buffer, containing any request that occurred during the broadcast cycle. These requests are sorted by deadline, and processed in this order. The amount of time spent processing broadcast data versus on-demand data is a tunable parameter in the system.

As mentioned above, mobile real-time database systems must contend with power constraints on the mobile devices that generate and access data. The work in [36] presents transaction management that reduces missed deadlines while balancing the amount of energy consumed by mobile hosts in the network. The system model in this work is a mobile ad-hoc network (MANET), in which only mobile hosts exist, not fixed hosts. Each mobile host can be in one of three energy modes: active (executing as usual), doze (CPU runs at lower rate and can receive messages), and sleep (CPU and communication are suspended). Small mobile hosts (SMH) have highly constrained memory, storage and power capabilities. Large mobile hosts (LMH) have more resources than small mobile hosts. Each LMH maintains a global schema that describes the status of the other LMHs. Each LMH broadcasts its ID, position and energy level to all other hosts. When and SMH starts a transaction, it is sent in its entirety to a LMH to be processed. To schedule the transactions, an energy-efficient modification of Least Slack Time scheduling is proposed that considers the problem of disconnection. Highest priority is given to the transaction with the smallest slack time. In case of equal slack time, higher priority is given to the transaction whose requestor has the lower energy level.

RTMonitor [64] is a real-time data monitoring system that uses mobile agents to adaptively monitor real-time data based on the urgency of the requests and on system workload. The target application for this system is traffic navigation. The agents us Adaptive PUSH OR PULL (APoP) to maintain the temporal consistency of the data. To manage the traffic system, local traffic graphs connect to neighboring local graphs through external nodes. The system consists of two types of agents: stationary and mobile. Stationary agents manage the traffic graphs. The mobile agents serve the client navigation requests. In the traffic system, traffic sensors collect traffic data for road segments and send the data whenever there is a significant change is a value. In the APoP approach used here, the monitoring period is defined based on the urgency of the request, the system status, and the dynamic properties of the traffic data. When a request is received, a GraphAgent is created to find the shortest path. This

is accomplished by computing a path on the local graph of the originating region, a virtual path to the destination region, and a local path from an external node to the destination in the destination region. To keep the calculated path up to date, the system initially uses a PUSH mode for data updates. When the workload at a server becomes heavy, some less critical requests will be switched to a PULL mode. The QueryAgent for a client generates a pull based on the slack time and the dynamic properties of the traffic data.

5.3 Web-based Real-time Data Services

Recently, the world wide web has offered a new venue for providing real-time data services. Many applications, such as program stock trading, require information services that can provide real-time data to widely distributed users. The main challenge in these applications is to provide timely access to fresh data, in the face of the highly dynamic environment of the web. Due to the unpredictable nature of web-based applications, most of the research in this area has been focused on providing QoS management for real-time data services.

In [103], a web-based content distribution service for industrial applications is presented. The service allows for remote monitoring of industrial devices that may be geographically distributed. It is based on an active web caching architecture that provides on-demand replication of dynamically changing web content, like industrial process state. The active web cache is made up of a standard web proxy cache that handles all static data, and an active server that handles one or more types of dynamic content. The approach in [103] is to trade-off data temporal consistency with timeliness, while keeping within specified inconsistency bounds. This trade-off is made through the formulation of a QoS optimization problem and a heuristic for QoS management. $MinC_i$ and $MaxC_i$ are the minimum and maximum allowed temporal consistency of a data item as defined by the publisher of the data. The consistency level of a data item is defined as:

$$QoS_i = (MaxC_i - CurrentC_i)/(MaxC_i - MinC_i) \tag{1}$$

and is used in the heuristic to manage QoS. The heuristic provides a mechanism for degrading the consistency level when the CPU utilization is above a given threshold.

A QoS management framework for web-based real-time data services, based on feedback control theory is presented in [113]. In this approach, a designer specifies desired behavior using performance metrics such as deadline miss ratio, data freshness, highest miss ratio and settling time, which are mapped to dynamic responses of control systems. The designer then determines a dynamic model of the real-time data services for performance control in order to define the relationship between the control input and the system performance. Using existing mathematical techniques of feedback control theory, the designer creates a feedback controller that provides analytic guarantees on the desired behavior. Using this technique, this work provides a solution to maintaining acceptable freshness in lazy replication. Each replicated data object has a primary copy that is updated periodically. Secondary copies of this data are created on sites where it is needed. These copies are updated by either a push or a pull model. At first, all secondary copies are updated at the same rate as the primary copy. When an overload occurs on a node, the replication can be adjusted by either performing on-demand copies, or delaying updates for some

copies.

The work in [6] handles QoS management in web-based data services by allowing imprecise computation. It describes two algorithms for feedback control with imprecision, which were described previously in Section 4.4.

5.4 Dissemination of Streaming/Dynamic Web Data

Recent studies have shown that an increasing fraction of the data on the world wide web is time-varying (i.e., changes frequently). Examples of such data include sports information, news, and financial information such as stock prices. The coherency requirements associated with a data item depends on the nature of the item and user tolerances. To illustrate, a user may be willing to receive sports and news information that may be out-of-sync by a few minutes with respect to the server, but may desire to have stronger coherency requirements for data items such as stock prices. A user who is interested in changes of more than a dollar for a particular stock price need not be notified of smaller intermediate changes.

An important issue in the dissemination of such time-varying web data is the maintenance of temporal coherency. In the case of servers adhering to the HTTP protocol, clients need to frequently pull the data based on the dynamics of the data and a user's coherency requirements. In contrast, servers that possess push capability maintain state information pertaining to clients and push only those changes that are of interest to a user. These two canonical techniques have complementary properties with respect to the level of temporal coherency maintained, communication overheads, state space overheads, and loss of coherency due to (server) failures.

Specifically a pull-based approach does not offer high fidelity when the data changes rapidly or when the coherency requirements are stringent. Moreover, the pull-based approach imposes a large communication overhead (in terms of the number of messages exchanged) when the number of clients is large. A push-based algorithm can offer high fidelity for rapidly changing data and/or stringent coherency requirements. However, it incurs a significant computational and state-space overhead resulting from a large number of open push connections. Moreover, the approach is less resilient to failures due to its stateful nature. These properties indicate that a push-based approach is suitable when a client requires its coherency requirements to be satisfied with a high fidelity, or when the communication overheads are the bottleneck. A pull-based approach is better suited to less frequently changing data or for less stringent coherency requirements, and when resilience to failures is important.

In [13] combinations of push- and pull-based techniques are proposed to achieve the best features of both approaches. The combined techniques tailor the dissemination of data from servers to clients based on (i) the capabilities and load at servers and proxies, and (ii) clients' coherency requirements.

Time-varying web data is frequently used for online decision making, and in many scenarios, such decision making involves multiple time-varying data items, possibly from multiple independent sources. Examples include a user tracking a portfolio of stocks and making decisions based on the net value of the portfolio. Observe that computing the overall value of the portfolio at any instant requires up-to-date values of each stock in the portfolio. In some scenarios, users might be holding these stocks in different (brokerage) accounts and might be using a third-party aggregator, such as yodlee.com, to provide them a unified view of all their investments. The third-

party aggregator assembles a unified view of the portfolio by periodically gathering and refreshing information from each individual account (source), all of which may be independent of one another. If the user is interested in certain events (such as a notification every time the value of the portfolio increases or decreases by $1000), then the aggregator needs to periodically and intelligently refresh the value of each stock price to determine when these user-specified thresholds are reached. More formally, the aggregator runs a continuous query over the portfolio by intelligently refreshing the components of the portfolio from their sources in order to determine when user-specified thresholds are exceeded. We refer to such queries as Continuous Threshold Queries (CTQ). Another example of CTQ would be an intersection of a busy thoroughfare where an increase in the traffic beyond a pre-specified limit triggers flow control measures. Much of the Web infrastructure assumes that it is difficult to cache time-varying data at a proxy, and consequently, requests for these data items as well as queries on these data items are handled directly by the server (data source). Such an approach has two important limitations. First, since continuous threshold queries are compute-intensive (due to the need to continuously evaluate the query condition), the server could become a bottleneck, and thereby, limit the scalability of the system. Second, a pure server-based approach makes CTQs on data items from multiple independent sources impossible (for example, the server-based approach is unfeasible in the case where a user specifies a query on stocks held in multiple independent accounts). Executing queries at a proxy eliminates both restrictions: the approach is scalable, since computations are offloaded from the server to proxies, and queries on data items from multiple sources become feasible. In addition, a proxy-based approach improves user response times. However, such an approach raises new research challenges. The key challenge is to ensure that the results of the query are no different from the case where the query is handled by the server (i.e., correctness of the results should be ensured). To do so, the proxy should ensure that cached values of time-varying data items are temporally coherent with the source.

6 Challenges and Research Agenda

The field of real-time database research has evolved a great deal over the relatively short time of its existence. In the early days of the 1980s, much of the research concentrated on examining how to add real-time support to traditional databases. Much of this work involved developing new scheduling and concurrency control techniques that extended existing techniques to allow RTDBs to provide timely transactions and temporally valid data. This has included techniques to relax traditional ACID properties of databases, and managing QoS provided by the database in order to allow for timing constraints to be met. More recently, the focus of RTDB research has been on applying the lessons learned from the earlier research towards providing real-time functionality to emerging applications that have data delivery and data management needs. In the most recent Real-Time and Embedded Applications Symposium (RTAS 2004), there was no session devoted to RTDBs, as there has been in the past. However, papers in real-time data dissemination, real-time data distribution, and real-time sensor networks, were presented in various other sessions. On the other hand, there was a session on RTDBs at the EuroMicro Conference on Real-Time Systems (ECRTS 2004) with papers on QoS issues in derived data and on embedded RTDBs in automobiles. This is an indication of the evolution of RTDB research, and the widening of the areas that are

covered in the field.

The future of RTDB research will depend on this continuing evolution. Research in this field should continue to follow cutting edge applications and apply existing techniques to them, as well as develop new ones when needed. It is also important that researchers keep up with new technology in traditional database research. As new approaches to solving database issues are developed, it is important to investigate how such innovations can apply to real-time applications. For example, a survey of papers in the current Conference on Very Large Databases (VLDB 2004) indicates that there is substantial research in XML, XQuery, database privacy, OLAP, data mining, data warehousing, query optimization and databases for the scientific grid. Research in RTDBs must keep up with the new technology in these areas and continue to ask the question, "Will this database research help solve real problems for RTDBs?" We can also ask the question, "Does it make sense to approach some of these issues from a real-time perspective?" For example, what does it mean to apply real-time technology to XML and XQuery? Also, are there useful applications for real-time data mining and data warehousing?

On a more practical note, it is important that RTDB research be widely applicable. Many academic papers mention various applications like programmed stock trading, medical patient monitoring, etc., to which their technologies will apply. However, there are very few real-time databases in use in such applications. In order for this research to continue, it must be shown to be practical and useful in real applications. There are currently several commercial solutions that provide some RTDB support [76, 128, 29, 27]. However, none of these employs the latest RTDB technology. Most are main memory database systems that provide "fast" response times. In order to further advance the state of the art research into deployed systems, it may be necessary for RTDB researchers to team with commercial vendors to develop systems that can be used by applications that require the kinds of technology that a RTDB can provide. Standards efforts may also be useful towards bringing the RTDB technology to the marketplace.

References

[1] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Transactions on Database Systems*, Vol. 17, No. 3, pp. 513-560, September 1992.

[2] R. Abbott and H. Garcia-Molina, "Scheduling I/O Requests with Deadlines: A Performance Evaluation," *Proceedings of the Real-Time Systems Symposium*, Dec. 1990.

[3] B. Adelberg, H. Garcia-Molina and B. Kao, "Applying Update Streams in a Soft Real-Time Database System," *Proceedings of the 1995 ACM SIGMOD*, pp. 245 - 256, 1995.

[4] B. Adelberg, B. Kao, and H. Garcia-Molina. Database Support for Efficiently Maintaining Derived Data. In *ETDB*, 1996.

[5] Q. N. Ahmed and S. V. Vrbsky. Triggered Updates for Temporal Consistency in Real-Time Databases. *Real-Time Systems Journal*, 19:209–243, 2000.

[6] M. Amirijoo, J. Hansson, and S. H. Son. Algorithms for managing QoS for real-time data services using imprecise computation. In *Proceedings of the Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, 2003.

[7] M. Amirijoo, J. Hansson, and S. H. Son. Error-driven QoS management in imprecise real-time databases. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, 2003.

[8] M. Amirijoo, J. Hansson, and S. H. Son. Specification and management of QoS in imprecise real-time databases. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, 2003.

[9] D. Bell and L. LaPadula. Secure Computer Systems: Unified Exposition and Multics Interpretation. *Tech. Report MTR-2997*, The Mitre Corp., March 1976.

[10] A. Bestavros and S. Braoudakis, "Timeliness via Speculation for RealTime Databases," in Proc. of 15th RealTime Systems Symp., December 1994.

[11] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. *IEEE Symposium on Security and Privacy*, Oakland, CA, pp 164-173, May 1996.

[12] S. Bhattacharya, H. Kim, S. Prabh, and T. Abdelzaher. Energy-Conserving Data Placement and Asynchronous Multicast in Wireless Sensor Networks. In *Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, 2003.

[13] Manish Bhide, Pavan Deolasse, Amol Katker, Ankur Panchbudhe, Krithi Ramamritham and Prashant Shenoy, Adaptive Push-Pull : Disseminating Dynamic Web Data, IEEE Transactions on Computers special issue on Quality of Service, June 2002, Vol 51, pp 652-668.

[14] B. Blum, P. Nagaraddi, A. Wood, T. Abdelzaher, S. H. Son, and J. A. Stankovic. An Entity Maintenance and Connection Service for Sensor Networks. In *Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA, 2003.

[15] P. Bonnet, J. Gehrke, and P. Seshadri. Querying the Physical World. *IEEE Personal Communication Magazine*, (7):10–15, Oct 2000.

[16] P. Bonnet, J. Gehrke, and P. Seshadri. Towards Sensor Database Systems. In *Proceedings of the 2nd International Conference on Mobile Data Management*, Hong Kong, 2001.

[17] E. Bosse, J. Roy, and S. Paradis. Modelling and Simulation in Support of Design of a Data Fusion System. *Information Fusion*, (1):77–87, Dec 2000.

[18] A.P. Buchmann, D.R. McCarthy, M. Chu, and U. Dayal, "Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control", *Proceedings of the Conference on Data Engineering*, 1989.

[19] M.J. Carey, R. Jauhari, and M. Livny, "Priority in DBMS Resource Scheduling", *Proceedings of the 15th VLDB Conference*, Aug 1989, pp. 397-410.

[20] W. Chang and M. Kam. Asynchronous Distributed Detection. *IEEE Transactions on Aerospace Electronic Systems*, pages 818–826, 1994.

[21] S. Chen, J. Stankovic, J. Kurose, and D. Towsley, "Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems", *Real-Time Systems*, Sept. 1991.

[22] D. Chen and A. K. Mok, "SRDE-Application of Data Similarity to Process Control," in *the 20th IEEE Real-Time Systems Symposium,* Phoenix, Arizona, December, 1999.

[23] Cougar Project. www.cs.cornell.edu/database/cougar.

[24] Dataman Project. www.cs.rutgers.edu/dataman.

[25] S. Davidson, I. Lee, and V. Wolfe, "A Protocol for Timed Atomic Commitment," in Proc. of 9th Intl. Conf. on Distributed Com puting Systems, 1989.

[26] W. Du and A. Elmagarmid. Quasi serializability: A Correctness Criterion for Global Concurrency Control in InterBase. In *Proceedings of the 15th International Conf. on Very Large Data Bases*, 1989.

[27] Empress. Real-Time Data Acquisition System: Empress Real-Time Data Collector. Whitepaper. www2.empress.com.

[28] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks.. *Proceedings of the 5th Annual International Conference on Mobile Computing and Networks*, Seattle, WA, 1999.

[29] eXtremeDB. Real-time Databases for Embedded Systems. Whitepaper. www.mcobject.com.

[30] J. Feng, F. Koushanfar, and M. Potkonjak. System-Architectures for Sensor Networks: Issues, Alternatives, and Directions. In *Proceedings of the 20th International Conference on Computer Design*, Freiburg, Germany, 2002.

[31] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems (3rd edition).* Addision Wesley, 1994.

[32] B. George and J. Haritsa. Secure Buffering in Firm Real-Time Database Systems. *Journal on Very Large Data Bases (VLDB)*, 8(3), pp 178-198, Feb 2000.

[33] M.C. Graham. "Issues in Real-Time Data Management", CMU/SEI-91-TR-17, July 1991.

[34] I. Greenberg et al. The Secure Alpha Study — Fianl Summary Report. *Technical Report*, CS Lab, SRI International, March 1993.

[35] N. Griffeth and A. Weinrib, "Scalability of a Real-Time Distributed Resource Counter", Proceedings of the Real-Time Systems Symposium, Orlando, Florida (December 1990).

[36] L. Gruenwald, S. Banik A Power-Aware Technique to Manage Real-Time Database Transactions in Mobile Ad-Hoc Networks. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications (DEXA'01)*, Munich, Germany, Sept. 2001.

[37] J.R. Haritsa, M.J. Carey and M. Livny, "On Being Optimistic about Real-Time Constraints," *Proceedings of ACM PODS*, 1990.

[38] J.R. Haritsa, M.J. Carey and M. Livny, "Dynamic Real-Time Optimistic Concurrency Control," *Proceedings of the Real-Time Systems Symposium*, Dec. 1990.

[39] J.R. Haritsa, M.J. Carey and M. Livny, "Earliest Deadline Scheduling for Real-Time Database Systems," *Proceedings of the Real-Time Systems Symposium*, Dec. 1991.

[40] J.R. Haritsa, M.J. Carey and M. Livny, "Data Access Scheduling in Firm Real-Time Database Systems," *The Journal of Real-Time Systems*, Vol. 4, No. 3, pp. 203-241, 1992.

[41] Jayant Haritsa, K. Ramamritham, and R. Gupta, The PROMPT Real-Time Commit Protocol, IEEE Trans. on Parallel and Distributed Systems, February 2000, pp. 160-181, Vol 11, Number 2.

[42] T. He, J. A. Stankovic, C. Lu, and T. Abdelzaher. SPEED: A Stateless Protocol for Real-Time Communication in Ad Hoc Sensor Networks. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, Providence, RI, 2003.

[43] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System Architecture Directions for Networked Sensors. *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[44] W. Hou, G. Ozsoyoglu, B. K. Taneja, "Processing Aggregate Relational Queries with Hard Time Constraints", *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, June 1989.

[45] J. Huang, J.A. Stankovic, D. Towsley and K. Ramamritham, "Experimental Evaluation of Real-Time Transaction Processing," *Proceedings of the Real-Time Systems Symposium*, Dec. 1989

[46] J. Huang, J.A. Stankovic, K. Ramamritham and D. Towsley, "Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes", *Proceedings of the Conference on Very Large Data Bases*, Sep 1991.

[47] J. Huang, J.A. Stankovic, K. Ramamritham, D. Towsley and B. Purimetla, "On Using Priority Inheritance in Real-Time Databases," **Special Issue** of *Real-Time Systems Journal*, Vol. 4. No. 3, September 1992.

[48] C. Jaikaeo, C. Srisathapornphat, and C-C Shen. Querying and Tasking in Sensor Networks. In *Proceedings of SPIE's 14th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Control (Digitization of the Battlespace V)*, Orlando, FL, 2000.

[49] D. Jayasimha, S. Ivengar, and R. Kashyap Information Integration and Synchronization in Distributed Sensor Networks. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1032–1043, Sep/Oct 1991.

[50] K. D. Kang, S. H. Son, and J. A. Stankovic. Managing Deadline Miss Ratio and Sensor Data Freshness in Real Databases. *IEEE Transactions on Knowledge and Data Engineering*, 2004. 16(7), July 2004.

[51] K. D. Kang, S. H. Son, J. A. Stankovic, and T. F. Abdelzaher. A QoS-Sensitive Approach for Timeliness and Freshness Guarantees in Real-Time Databases. In *the 14th Euromicro Conference on Real-Time Systems*, June 2002.

[52] E. Kayan, O. Ulusoy An Evaluation of Real-Time Transaction Management Issues in Mobile Database Systems. *The Computer Journal* (Special Issue on Mobile Computing), vol. 42, no. 6, 1999.

[53] S. Kim, S. H. Son, J. A. Stankovic, S. Li, and Y. Choi. SAFE: A Data Dissemination Protocol for Periodic Updates in Sensor Networks. In *IEEE Workshop on Data Distribution for Real-Time Systems (DDRTS)* , Providence, RI, May 2003.

[54] S. Kim, S. Son, and J. Stankovic. Performance Evaluation on a Real-Time Database. In *IEEE Real-Time Technology and Applications Symposium*, 2002.

[55] W. Kim and J. Srivastava, "Enhancing Real-Time DBMS Performance with Multi-version Data and Priority-Based Disk Scheduling", *Proceedings of the Real-Time Systems Symposium*, Dec 1991, pp. 222-231.

[56] H. F. Korth, Soparkar, Silberschatz, A. "Triggered Real-Time databases with consistency constraints", *Proceedings of the Conference on Very Large Data Bases*, 1990.

[57] T. Kuo and A. K. Mok, "Real-Time Data Semantics and Similarity-Based Concurrency Control," *IEEE 13th Real-Time Systems Symposium* , December 1992.

[58] T. Kuo and A. K. Mok, "SSP: a Semantics-Based Protocol for Real-Time Data Access," *IEEE 14th Real-Time Systems Symposium* , December 1993.

[59] T. Kuo and A. K. Mok, "Real-Time Data Semantics and Similarity-Based Concurrency Control," *IEEE Transactions on Computers*, 49(11):1241-1254 (2000).

[60] S. Ho, T. Kuo, and A. K. Mok, "Similarity-Based Load Adjustment for Static Real-Time Transaction Systems," *18th Real-Time Systems Symposium*, 1997.

[61] A. Labrinidis and N. Roussopoulos. Adaptive WebView Materialization. In *the Fourth International Workshop on the Web and Databases, held in conjunction with ACM SIGMOD*, May 2001.

[62] A. Labrinidis and N. Roussopoulos. Update Propagation Strategies for Improving the Quality of Data on the Web. In *the 27th International Conference on Very Large Data Bases (VLDB'01)*, Rome, Italy, September 2001.

[63] K. Lam, Concurrency Control in Distributed RealTime Database Systems, Ph.D. thesis, City Univ. of Hong Kong, Oc tober 1994.

[64] K-Y. Lam, A. Kwan, K. Ramamritham RTMonitor: Real-Time Data Monitoring Using Mobile Agent Technologies. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[65] K-Y. Lam, T-W. Kuo, W-H. Tsang, G. Law Concurrency Control in Mobile Distributed Real-Time Database Systems. *Information Systems*, vol. 25, no. 4, June 2000, pp. 261-322.

[66] K-Y. Lam, G. Li, T-W. Kuo A Multi-Version Data Model for Executing Real-Time Transactions in a Mobile Environment. In *Proceedings of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access*, 2001.

[67] B. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10), pp 613-615, October 1973.

[68] H-W. Leung, J. Yuen, K-Y. Lam, E. Chan Enhanced Multi-Version Data Broadcast Schemes for Time-Constrained Mobile Computing Systems. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA'02)*, Sept. 2002.

[69] S. Li, Y. Lin, S. H. Son, J. Stankovic, and Y. Wei. Event Detection Services using Data Service Middleware in Distributed Sensor Networks. *Telecommunication Systems*, (26):351–368, 2004.

[70] K. J. Lin, S. Natarajan, and J. W. S. Liu. Imprecise Results: Utilizing Partial Computations in Real-Time Systems. In *Real-Time System Symposium*, December 1987.

[71] Y. Lin and S.H. Son, "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order," *Proceedings of the Real-Time Systems Symposium*, Dec. 1990.

[72] C. L. Liu, and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, 20(1), 1973.

[73] J. Liu, K. Lin, W. Shih, A. Yu, J. Chung, and W. Zhao, "Algorithms for Scheduling Imprecise Computation", *IEEE Computer*, Vol. 24, No. 5, May 1991.

[74] J. W. S. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung. Imprecise computations. *Proceedings of the IEEE*, 82, Jan 1994.

[75] Doug Locke, "Real-Time Databases: Real-World Requirements," in *Real-Time Database Systems: Issues and Applications,* edited by Azer Bestavros, Kwei-Jay Lin and Sang H. Son, Kluwer Academic Publishers, pp. 83-91, 1997.

[76] Lockheed Martin Corporation http://www.lmco.com, July 1998.

[77] C. Lu, T. Abdelzaber, J. Stankovic, and S. Son. A feedback control approach for guaranteeing relative delays in web servers. In *Proc. 7th IEEE Real-Time Technology and Applications Symposium (RTAS 01)*, pages 51–62, Taipei, Taiwan, May 2001. IEEE.

[78] C. Lu, B. Blum, T. Abdelzaher, J. A. Stankovic, and T. He. RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks. In *Proceedings of the 8th IEEE Real-Time Technology and Applications Symposium*, San Jose, CA, 2002.

[79] C. Lu, J. Stankovic, T. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance Specifications and Metrics for Adaptive Real-Time Systems. In *Real-Time Systems Symposium*, Orlando, Florida, November 2000.

[80] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms. *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing,*, 2001. To appear.

[81] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real-time Systems*, 23(1/2), July/September 2002. Special Issue on Control-Theoretical Approaches to Real-Time Computing.

[82] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *Proceedings of the 18th International Conference on Data Engeneering*, San Jose, CA, 2002.

[83] S. Madden and M. J. Franklin. Fjording The Stream: An Architecture for Queries over Streaming Sensor Data. In *Proceedings of the 18th International Conference on Data Engeneering*, San Jose, CA, 2002.

[84] F. Mattern, K. Rőmer, O. Kasten. Middleware Challenges for Wireless Sensor Networks. *ACM SIGMOBILE Mobile Computing and Communication Review (MC2R)*, 2002.

[85] I. Mostowitz, S. Greenwald, and M. Kang. An Analysis of Timed Z-Channel. *IEEE Symposium on Security and Privacy*, Oakland, CA, pp 2-11, May 1996.

[86] P. R. Murphy. Dempster-Shafer Theory for Sensor Fusion in Autonomous Mobile Robots. *IEEE Transactions on Robotics and Automation*, 14(2):197–206, Apr 1999.

[87] P. O'Neil, K. Ramamritham, and C. Pu, "Towards Predictable Transaction Executions in Real-Time Database Systems", Technical Report 92-35, University of Massachusetts, August, 1992.

[88] G. Ozsoyoglu and W.-C. Hou. Research in Time- and Error-Constrained Database Query Processing. In *Workshop on Real-Time Operating Systems and Software*, May 1990.

[89] H. Pang, M.J. Carey and M. Livny, "Multiclass Query Scheduling in Real-Time Database Systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 4, August 1995.

[90] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance managment. *Journal of Real-time Systems*, 23(1/2), July/September 2002. Special Issue on Control-Theoretical Approaches to Real-Time Computing.

[91] C. Park, S. Park, and S. H. Son. Multiversion Locking Protocol with Freezing for Secure Real-Time Database Systems. *IEEE Transactions on Knowledge and Data Engineering*, 14(5), pp 1141-1154, Sept 2002.

[92] C. L. Phillips and H. T. Nagle. *Digital Control System Analysis and Design (3rd edition).* Prentice Hall, 1995.

[93] C. Pu and A. Leff. Replica Control in Distributed Systems: As Asynchronous Approach. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1991.

[94] H. Qi, X. Wang, S. S. Iyengar, and K. Chakrabarty. Multisensor Data Fusion in Distributed Sensor Networks Using Mobile Agents. In *Proceedings of 5th International Conference on Information Fusion*, Annapolis, MD, 2001.

[95] K. Ramamritham, "Real-Time Databases," *Distributed and Parallel Databases* 1(1993), pp. 199-226, 1993.

[96] K. Ramamritham, "Where Do Deadlines Come from and Where Do They Go?" *Journal of Database Management*, Spring, 1996.

[97] R. Rastogi, S. Seshadri, J. Bohannon, D. Leinbaugh, A. Silberschatz, and S. Sudarshan, Improving Predictability of Transaction Execution Times in Real-Time Databases. *Real-Time Systems Journal* 2000,19(3) : pp. 205-208

[98] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, S. Shenker, L. Yin, and F. Yu. Data-Centric Storage in Sensornets. In *Proceedings of the 1st Workshop on Sensor Networks and Applications*, Atlantic, GA, 2002.

[99] D. Rosu, K. Schwan, S. Yalmanchili, and R. Jha. On Adaptive Resource Allocation for Complex Real-Time Applications. In *IEEE Real-Time Systems Symposium*, Dec 1997.

[100] V. N. S. Samarasooriya and P. K. Varshney. A Sequential Approach to Asynchronous Decision Fusion. *Optical Engineering*, 35(3):625–633, Mar 1996.

[101] V. N. S. Samarasooriya and P. K. Varshney. A Fuzzy Modeling Approach to Decision Fusion under Uncertainty. *Fuzzy Sets and Systems*, 114(1):59–69, Aug 2000.

[102] SCADDS: Scalable Coordination Architectures for Deeply Distributed Systems. www.isi.edu/scadds.

[103] S. Sebastine, K-D. Kang, T. Abdelzaher, S.H. Son A Scalable Web-Based Real-Time Information Distribution Service for Industrial Applications. In *Proceedings of the 27th Annual Conference of IEEE Industrial Electronics Society*, Denver, Co., Dec. 2001.

[104] L. Sha, R. Rajkumar and J.P. Lehoczky, "Concurrency Control for Distributed Real-Time Databases," *ACM SIGMOD Record*, March 1988.

[105] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", *IEEE Transactions on Computers*, 39(9), pp. 1175-1185, 1990.

[106] C-C Shen, C. Srisathapornphat, and C. Jaikaeo. Sensor Information Networking Architecture and Applications. *IEEE Personel Communication Magazine*, 8(4):52–59, Aug 2001.

[107] Raju Sivasankaran, Krithi Ramamritham and J.A. Stankovic, System Failure and Recovery, in Real-Time Database Systems: Architecture and Techniques, K-Y Lam and T-W Kuo (ed.), Kluwer Academic Publishers, 2000, pp.109-124.

[108] R.M. Sivasankaran, J.A. Stankovic, D. Towsley, B. Purimetla and K. Ramamritham, "Priority Assignment in Real-Time Active Databases," *The International Journal on Very Large Data Bases*, Vol. 5, No. 1, January 1996.

[109] Smart Messages Project. discolab.rutgers.edu/sm.

[110] L. Shu, S. H. Son, and J. Stankovic. Achieving Bounded and Predictable Recovery using Real-Time Logging. *The Computer Journal*, 47 (4), pp 373-394, May 2004.

[111] S. H. Son, R. David, and C. Chaney. Design and Analysis of an Adaptive Policy for Secure Real-Time Locking Protocol. *Journal of Information Sciences*, 99, pp 101-135, June 1997.

[112] S. H. Son, C. Chaney, and N. Thomlinson. Partial Security Policies to Support Timeliness in Secure Real-Time Databases. *IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 136-147, May 1998.

[113] S.H. Son, K-D. Kang QoS Management in Web-based Real-Time Data Services. In *Proceedings of the 4th IEEE Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'02)*, Newport Beach, CA., June 2002.

[114] S.H. Son, Y. Lin, and R. P. Cook, "Concurrency Control in Real-Time Database Systems", in *Foundations of Real-Time Computing: Scheduling and Resource Management*, edited by Andre van Tilborg and Gary Koob, Kluwer Academic Publishers, pp. 185-202, 1991.

[115] Son, S., "Using Replication for High Performance Database Support in Distributed RealTime Systems," Proceedings of the 8th IEEE RealTime Systems Symposium, pp. 7986, 1987.

[116] Son, S., and Kouloumbis, S., "A Real-Time Synchronization Scheme for Replicated Data in Distributed Database Systems," Information Systems, 18(6), 1993.

[117] S. H. Son, R. Mukkamala, and R. David Integrating Security and Real-Time Requirements using Covert Channel Capacity. *IEEE Transactions on Knowledge and Data Engineering*, 12(6), pp 865-879, Dec. 2000.

[118] Son, S., and Zhang, F., "RealTime Replication Control for Distributed Database Systems: Algorithms and Their Perfor mance," 4th International Conference on Database Systems for Advanced Applications, Singapore, April, 1995.

[119] S. H. Son, R. Zimmerman, and J. Hansson. An Adaptable Security Manager for Real-Time Transactions. In *Euromicro Conference on Real-Time Systems*, pages 63–70, Stockholm, Sweden, June 2000.

[120] X. Song and J.W.S. Liu, "How Well Can Data Temporal Consistency be Maintained?" *Proceedings of the IEEE Symposium on Computer-Aided Control Systems Design*, 1992.

[121] X. Song and J. W. S. Liu, "Maintaining Temporal Consistency: Pessimistic vs. Optimistic Concurrency Control," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 5, pp. 786-796, October 1995.

[122] N. Soparkar et al, "Adaptive Commitment for Real-Time Distributed Transactions," Tech. Rep. TR9215, Dept. of Computer Science, Univ. of TexasAustin, 1992.

[123] J. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu. Feedback control scheduling in distributed real-time systems. In *Proc. 22nd IEEE Real-Time Systems Symposium (RTSS 01)*, pages 59–72, London, UK, Dec 2001. IEEE.

[124] J.A. Stankovic, K. Ramamritham, and D. Towsley, "Scheduling in Real-Time Transaction Systems," in *Foundations of Real-Time Computing: Scheduling and Resource Management*, edited by Andre van Tilborg and Gary Koob, Kluwer Academic Publishers, pp. 157-184, 1991.

[125] J.A. Stankovic, S. Son, J. Hansson, "Misconceptions About Real-Time Databases," IEEE Computer, Vol. 32, No. 6, pp. 29-36, June 1999.

[126] D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proc. 3rd Symposium on Operating Systems Design and Implementation (OSDI 99)*, pages 145–158, Berkeley, CA, February 22–25 1999.

[127] B. Thuraisingham and W. Ford. Security Constraint Processing in a Multilevel Secure Distributed Database Management System. *IEEE Trans. on Knowledge and Data Engineering*, 7(2), April 1995.

[128] TimesTen. TimesTen Real-Time Event Processing System: Product and Technology Overview. Whitepaper. www.timesten.com.

[129] B. Timmerman, A Security Model for Dynamic Adaptive Traffic Masking. *New Security Paradigms Workshop*, pp 1-25, Sept. 1997.

[130] T. C. Ting. How Secure is Secure?. *IFIP WG 11.3 Working conference on Database Security*, Opening Remarks.

[131] M. R. Tremblay and M. R. Cutkosky. Using Sensor Fusion and Contextual Information to Perform Event Detection during a Phase-Based Manipulation Task. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, 1995.

[132] O. Ulusoy, "Processing RealTime Transactions in a Replicated Database System," Intl. Journal of Distributed and Parallel Databases, vol. 2, no. 4, 1994.

[133] S. Vrbsky. *APPROXIMATE: A Query Processor that Produces Monotonically Improving Approximate Answers.* PhD thesis, University of Illinois at Urbana-Champaign, 1993.

[134] S. V. Vrbsky and K.J. Lin. "Recovering Imprecise Computations with Real-Time Constraints", *Proceedings of the Seventh Symp. on Reliable Distributed Systems*, October 1988, pp. 185-193.

[135] Ahmed and S. Vrbsky, Triggered Updates for Temporal Consistency in Real-Time Databases. *Real-Time Systems Journal* 2000,19(3) : pp. 205-208

[136] Y. Wei, S. Son, J. Stankovic, and K.D. Kang. Qos management in replicated real-time databases. In *24th IEEE Real-Time Systems Symposium (RTSS 2003)*, Dec 2003.

[137] Y. Wei, A. Aslinger, S. Son, and J. Stankovic, ORDER: A dynamic replication algorithm for periodic transactions in distributed real-time databases. In *10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA 2004)*, Aug. 2004.

[138] J. Wray, An Analysis of Covert Timing Channels. *IEEE Symposium on Security and Privacy*, Oakland, CA, pp 2-7, May 1991.

[139] M. Xiong, R. M. Sivasankaran, J. Stankovic, K. Ramamritham, and D. Towsley, "Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics," *IEEE 17th Real-Time Systems Symposium,* pp. 240-251, December 1996.

[140] M. Xiong and K. Ramamritham, "Deriving Deadlines and Periods for Real-Time Update Transactions," in *Proc. 20th IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December, 1999.

[141] M. Xiong, K. Ramamritham, J. Haritsa and J. A.Stankovic, Mirror: A State-Conscious Concurrency Control Protocol for Replicated Real-Time Databases Information Systems: An International Journal, 2002,27(4):pp.277-297.

[142] M. Xiong, K. Ramamritham, J. A. Stankovic, Don Towsley and R. Sivasankaran Scheduling Transactions with Temporal Constraints:Exploiting Data Semantics, IEEE Transactions on Knowledge and Data Engineering, Sep/Oct 2002, Vol 14, No:5, pp 1155-1166.

[143] M. Xiong and K. Ramamritham, Deriving Deadlines and Periods for Real-Time Update Transactions, IEEE Transactions on Computers, Vol 53, No.5, pp 567 - 583, May 2004.

[144] S. Yan, S. Wang, and Z. Dou. An Energy Model in Fire Detection and Integrated Analysis on False Alarms. In *Proceedings of the 12th International Conference on Automatic Fire Detection*, Gaithersburg, MD, 2001.

[145] Y. Yoon, Transaction Scheduling and Commit Processing for RealTime Distributed Database Systems, Ph.D. thesis, Korea Advanced Institute of Science and Technology, May 1994.

[146] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*, Alberta, Canada, 1998.

[147] H. Zhou, and F. Jahanian, "Real-Time Primary-Backup (RTPB) Replication with Temporal Consistency Guarantees," *Proceedings of ICDCS*, May, 1998.

[148] S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.

[149] M. Zurko and R. Simon. User-Centered Security. *New Security Paradigms Workshop*, Lake Arrowhead, CA, pp 27-33, Sept. 1996.