John A. Stankovic and Sang Hyuk Son
University of Virginia, Charlottesville

Jorgen Hansson
University of Skovde, Sweden

Misconceptions About Real-Time Databases

Some database users think "real-time" databases just need to be fast and that conventional databases are adequate for real-time applications. Real-time database designers don't agree.

atabases have become an integral part of many computer systems—ranging from complex systems that control air traffic, plant operations, and stock market transactions to general-purpose computing systems that run audio and video applications. Increasingly, computer systems—even general-purpose systems—are requiring real-time support, so it's not surprising to hear more about real-time databases.

Unfortunately, there are many misconceptions about the real-time aspects of databases. Ironically, the state of confusion that exists today about real-time databases parallels the confusion that existed a decade ago surrounding the differences between real-time and general-purpose computing.[1] We believe that a careful definition of real-time databases will help dispel these misconceptions and will encourage research efforts similar to those that have advanced real-time systems over this past decade.

REAL-TIME DATABASES: SOME DEFINITIONS

We must first note that a system using real-time data, such as sensor data, does not in itself constitute a real-time database system. Because a real-time database is by definition a database system, it has queries, schemas, transactions, commit protocols, concurrency control support, and storage management.

In a real-time database system, timing constraints are associated with transactions, and data are valid for specific time intervals.[2,3] The transaction timing constraints can be completion deadlines, start times, periodic invocations, and so on. It is not necessary that every transaction have a timing constraint, only that some do.

In addition to transaction timing requirements, data has *time semantics* as well. Data such as sensor data, stock market prices, and locations of moving objects all have semantics indicating that the recorded values are valid only for a certain time interval. A real-time database makes this validity interval explicit as part of its database schema.

We can define transaction *correctness* as a transaction meeting its timing constraints and using data that is absolutely and relatively timing-consistent. *Absolute time consistency* means that individual data items used by a transaction are still temporally valid and reflect the true state of the world to an acceptable degree of accuracy. *Relative time consistency* means that multiple data items used by a transaction are updated (sensed) within a specified time interval of each other. For example, if a transaction uses temperature and pressure data to make a decision regarding a chemical process, these two data values must correlate closely in time or the computation will likely make no sense.

0018-9162/99/$10.00 © 1999 IEEE

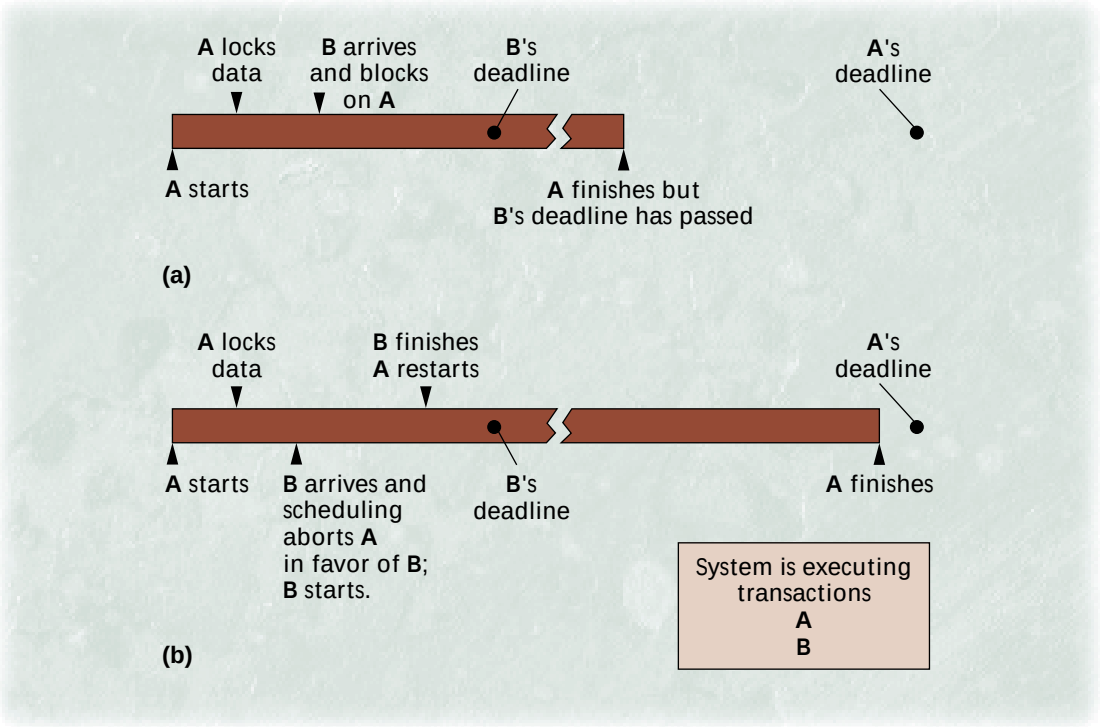

Figure 1. Processing of two transactions using (a) conventional database protocols and (b) time-cognizant protocols.

Using these definitions, we can better explain some common misconceptions about real-time databases.

SOME MISCONCEPTIONS ABOUT REAL-TIME DATABASES

We present nine common misconceptions about real-time databases. The first three mistakenly argue that real-time systems are synonymous with speed. The next three argue that current database technology can be used in real-time database systems. These two groups of misconceptions are based on some common assumptions, but it is instructive to distinguish between them. Finally, we examine three fallacies about real-time database properties: temporality, predictability, and specialization.

Part of the confusion over real-time database systems stems from the work of two communities coming together. Mainstream database researchers and users usually do not have much experience with real-time issues, whereas the real-time system community has dealt primarily with real-time data derived from sensors and has not associated real-time issues with databases.

Real-time means speed

Mainstream database users usually do not have any experience with real-time issues and think it is acceptable simply to make a commercial database management system run fast.

Hardware advances will address real-time database requirements. Technology will exploit parallel processors to improve system throughput, but this does not mean that such systems will automatically meet timing constraints. In fact, the increased size and complexity of databases and hardware will make it more difficult to meet timing constraints or to show that such constraints will be met. Hardware alone cannot ensure that transactions will be scheduled properly to meet their deadlines, nor can it ensure that the data is still temporally valid. A transaction that uses obsolete data more quickly is still incorrect.

Advanced database technology will address real-time database requirements. Some database designers claim that better buffering, faster commit protocols, and novel query-processing techniques will speed up databases sufficiently for use in real-time systems. While these techniques help, they can't guarantee either the required deadlines or temporal validity of the data. Required advances in database technology include time-cognizant protocols for concurrency control, commit processing, transaction scheduling, and logging and recovery. Ample evidence now exists that such protocols are considerably better at supporting real-time transaction and data correctness than standard database protocols that simply go fast.[4-8]

Real-time computing is fast computing. Fast computing aims to minimize the average response time of a set of transactions. In contrast, real-time databases aim to meet the timing constraints and data-validity requirements of individual transactions and also keep the database current via proper update rates. To do this, we need time-cognizant protocols.

Figure 1 illustrates how a time-cognizant protocol could improve the processing of two transactions, A and B. Conventional database protocols, which generally schedule transactions on a first-come, first-serve basis, will let A lock the data and complete, allowing A to meet its deadline. B, on the other hand, will miss its deadline because A's lock on the data prevents B from starting early enough.

In contrast, a real-time database with time-cognizant protocols would preempt transaction A and transfer data control to B because B's deadline is ear-

lier. Transaction A would regain control after B completes, and both transactions would meet their deadlines in this example.

Current database technology can solve real-time problems

Mainstream database designers also sometimes believe they can shoehorn real-time principles into traditional databases.

Traditional databases can handle realtime. This is tricky. A current database system can define a field for every relation (object) that contains the validity interval of that data. Then the transaction itself can check these fields to ensure absolute and relative validity. However, this means that every transaction must include this capability itself instead of having the system support it. Furthermore, you can modify the system to run some form of earliest deadline scheduling by controlling the priority of each transaction.

By adding these two features, however, designers are in fact moving toward a real-time database system. If transactions have such constraints, it is more efficient to build them into the system than to force fit a typical database into this set of capabilities. Furthermore, if you actually do this force fitting, you now have a real-time database system that will very likely not be as efficient as one developed from the ground up with these capabilities. After all, all algorithms are programmable on a Turing machine, but few people would advocate using a Turing machine to build real systems.

Placing a conventional database in main memory is sufficient. Some nonreal-time database designers argue that placing a conventional database in main memory is a viable way to gain performance and thereby make it suitable for real-time systems. Although main-memory resident databases do eliminate disk delays, conventional databases still have many additional sources of unpredictability—such as delays due to blocking on locks, transaction scheduling, stolen processing time to handle external interrupts, and so on—that prevent time constraints from being ensured. Again, increases in performance cannot completely make up for the lack of time-cognizant protocols in conventional database systems.

A real-time database must reside totally in main memory. The previous misconception is the view held by some nonreal-time database designers. Real-time designers often hold the same view, but from a more dogmatic perspective: You must place the database in main memory. This is not correct either. The primary reasons for placing data in main memory are to increase speed and avoid the unpredictable seek and rotational delays introduced by disks.

The primary issue here is I/O. In most systems, I/O requests are scheduled to minimize average response time, maximize throughput, or maintain fairness. Typical disk scheduling algorithms for this type of disk scheduling are First-Come-First-Served (FCFS), Shortest-Seek-Time-First (SSTF), and the elevator algorithm SCAN. Typically, a database transaction performs a sequence of database read operations, computations, and then writes the data back to the database. However, since the deadline and the importance of the transaction are not considered when disk requests are scheduled, the timeliness of the transaction is jeopardized.

Mainstream database designers also sometimes believe they can shoehorn real-time principles into traditional databases.

In the same way traditional CPU scheduling algorithms have been shown to be inappropriate for real-time systems, the use of nontime-cognizant disk scheduling algorithms are inappropriate for scheduling disk requests. Disk scheduling algorithms that combine a scan and deadline requirement work considerably better than conventional algorithms.[9] It is likely that some combined solution will prevail where critical data is placed and pinned in (nonvolatile) main memory and less critical data is stored on the disk using time-cognizant disk scheduling.

Fallacies about real-time databases

There are also some generic misconceptions about the properties of real-time databases, including temporality, predictability, and specialization.

A temporal database is a real-time database. Although temporal databases and real-time databases both support time-specific computation, they support different aspects of time. A temporal database supports those aspects of time associated with information—for example, time-variant information such as stock quotes—whereas a real-time database tries to satisfy timing constraints associated with operational aspects of the database.

In the context of databases, two temporal dimensions are of particular interest:

- *valid time*, or the time at which a fact is true in reality, and
- *transaction time*, or the time during which a fact is present in the database as stored data.[10]

These two dimensions are in general orthogonal, although there could be some application-dependent correlations between them.

Consider the difference between a temporal database and a real-time database in the following example. The military rank of Beetle Bailey can be specified in a temporal database as that of private between January 1, 1998 and June 30, 1999, at which time he will be promoted. It only states the timing fact that is believed to be true, regardless of when that information was entered. A real-time database, on the other hand, might

While each real-time database application may have different timing constraints, specialized database systems need not be developed from scratch for each application.

contain Beetle Bailey's blood pressure, which is valid for only a short time after it was measured.

In most real-time databases, the static timing facts found in temporal databases are not a primary concern. In a real-time database, the valid time is specified according to the semantics of the external object in the real world. When a value is entered into the database, its valid time specifies that the value can be assumed to represent the actual value (absolute time consistency). If the value of sensor data was inserted into the database at time T and its valid time interval is t, then the value must be updated within time $T + t$; if it is not updated within that time, the value becomes stale and useless, or even dangerous. Current temporal database research does not pursue operational timing constraints such as maintaining correlation to real-time events in the real world and meeting deadlines.

Because of their different objectives, real-time and temporal databases use different policies and mechanisms to resolve data and resource conflicts. Since meeting timing constraints is essential in certain safety-critical database applications, a real-time database needs to provide a range of transaction correctness criteria that relax ACID (atomicity, consistency, isolation, durability) properties. However, such an approach is generally not acceptable in temporal databases. Temporal databases, along with other conventional databases, attempt to be fair while maximizing resource utilization. In real-time databases, timely execution of transactions is more important, and fairness and resource utilization are secondary considerations.

Real-time database systems can't make guarantees or achieve predictability. Some argue that real-time databases cannot achieve predictability due, in part, to the complexity of making accurate—and not overly pessimistic—estimates of transaction execution times. This complexity results because database systems have a number of sources of unpredictability.[3] Because a transaction's execution sequence is heavily dependent on data values, predictability can be adversely affected by data and resource conflicts, dynamic paging and I/O, and transaction aborts resulting in rollbacks and restarts. As discussed earlier, placing the database in main memory or adopting time-cognizant protocols for scheduling disk requests and managing memory buffers would help alleviate the unpredictability due to disk delays.

Although it is difficult to evaluate the data-dependence of general-purpose transactions, many real-time transactions are

- specialized (updating periodic sensor data, for example),
- fixed (using the same type and amount of data each time), and/or
- prewritten and evaluated offline (as in canned transactions).

This set of features enables associated protocols to utilize such information and improve predictability.

Data conflicts may cause general-purpose transactions to roll back and restart, increasing execution times and, in the worst case, causing not only the transaction to miss its deadline, but also jeopardizing the timeliness of other transactions requesting resources. In real-time systems, on the other hand, the set of transactions is normally well known and we can estimate needed resources, such as execution times and required data. With this information, we can thus minimize and bound the number of data conflicts and transaction restarts.

Although we have made much progress in improving predictability, this issue is still very much an open research question.

A real-time database is a specialized database. While each real-time database application may have different timing constraints, specialized database systems need not be developed from scratch for each application. By analogy, such an assertion would be tantamount to saying that any real-time application needs its own specialized real-time operating system, since its resource requirements and scheduling policies are different from others. Although specific timing requirements can vary among applications, each application needs database support for specifying and enforcing its requirements.

At the same time, conventional database systems cannot be used for real-time applications simply by adding a few functional improvements. Since support for timing constraints deals with the lowest level database access mechanisms, the overall architecture of database systems must become time cognizant for the very same reasons that certain time-critical applications need real-time operating systems instead of conventional operating systems. However, different real-time operating systems don't have to be developed for each application.

RESEARCH CHALLENGES

While a significant amount of real-time database research has been done,[11] this field is still in its infancy. We now discuss several key research challenges.

System support

Database transactions differ from traditional tasks (processes) in several ways. One main difference involves the process of obtaining good estimates of worst-case execution time. Other differences include developing alternative correctness criteria, improving buffer management, and resolving issues in transaction abort and recovery.

Determining worst-case execution times. Obtaining useful worst-case execution times for transactions is

a complex issue because transactions normally involve multiple resources, such as CPU, I/O, buffers, and data. It is also difficult to assess the impact of blocking on transaction response times. While a concurrency control protocol aims to ensure database consistency by controlling the interleaved execution of concurrent transactions, it too affects the transaction's response time.

Furthermore, transaction execution times usually depend heavily on the volume of data and the data values read from the database. Thus, to enforce the timeliness of transactions and maintain database consistency, scheduling algorithms must consider both hardware resources and the scheduling of data resources. We must therefore determine how to better integrate concurrency control and the scheduling of transactions.

Correctness criteria. In conventional databases, *serializability* is the primary correctness criterion for transaction schedules: The result produced by the interleaved execution of a set of transactions should be identical to one produced by executing the transactions in some serial order.

Although serializable schedules ensure correctness, under many circumstances the cost of enforcing serializability in real-time database systems is sometimes too high. This is especially true for the class of real-time applications where timeliness is essential. In this case, producing a useful result on time with a nonserializable schedule is better than producing a result too late with a serializable schedule. A key challenge is to define new and alternative criteria for database correctness and develop methods that trade serializability for timeliness.

Buffer management. In conventional systems, buffer management aims to reduce transaction response times. Buffer items in these systems are normally allocated and replaced based on the transactions' reference behavior. However, this approach can degrade performance in a real-time system. For example, replacing buffer slots referenced by currently executing (and not yet committed) transactions can delay the completion of transactions or even cause missed deadlines.

Buffer management policy must also account for other semantics, such as periodic transactions. A poor buffer management policy might allow pages to be replaced just prior to the transaction's next periodic invocation. The challenge here is to develop policies that consider the transaction's importance and its temporal requirements as well as enforce predictability.

Transaction abort and recovery. Transaction abort and recovery consumes valuable processing time and may affect other currently executing transactions. Establishing the time at which recovery should be performed therefore requires careful consideration.

In systems with lock-based concurrency control protocols, we want to avoid prolonged waiting on the part of transactions accessing resources locked by the recovering transaction; hence, locks should be released as early as possible.

Optimistic protocols, on the other hand, perform conflict detection and resolution at the end of the transaction. After the transaction has executed, the concurrency control manager is notified and checks for data conflicts. If it detects a conflict, it aborts and then restarts the transaction. A transaction may be restarted a number of times before it can commit successfully. This can cause problems in a real-time database system because the increased processing time due to transaction restarts can make the transaction late and can jeopardize the timeliness of other transactions as well. Resolving these issues in transaction abort and recovery are key research challenges.

Obtaining useful worst-case execution times for transactions is a complex issue because transactions normally involve multiple resources.

Distributed and global systems

Whereas many real-time systems perform a set of tasks that are well understood at design time and thus permit static solutions, large real-time systems—such as air traffic control, autonomous vehicles, and missile control systems—typically operate for long periods in complex nondeterministic and fault-inducing environments under severe time constraints. This gives rise to the need for dynamic solutions and robust real-time databases delivering real-time performance. Furthermore, because these systems are highly distributed, we need to develop distributed protocols for open real-time systems and applications.

Composition. Composition is the process of combining modules and/or subsystems to achieve some new capability. Composition can be done offline at design time or online as a result of the system reacting to new conditions. Composition has long been recognized as a key research issue for many systems, including real-time database systems. However, we have tended to focus largely on functional composition of a single system or application.

For today's distributed and global systems, we need to extend composition across multiple interacting domains: function, time, fault tolerance, and security. Today, most work focuses only on functional composition. Furthermore, both offline and online solutions are required. Online composition must dynamically create new system actions by combining functional, timing-aware, fault-tolerant, and secure components, within its own timing and fault tolerance requirements.

Finally, the results should permit verification. Verifiable results will lead to adaptive high-performance, fault-tolerant embedded systems that dynamically address real-time constraints. Such systems will provide both a priori acceptable system-level perfor-

Developing an effective architecture that supports general-purpose, open real-time systems and applications is another challenge facing the real-time community.

mance guarantees and graceful degradation in the presence of failures, time constraints, and database access. How low-level real-time techniques and real-time database technology interact is a critical issue.

The distributed nature of the systems also gives rise to new database research. Issues to be investigated include developing distributed real-time concurrency control, commit protocols, and transaction scheduling; meeting end-to-end timing constraints when database accesses are involved; supporting replication of data in real time; and ensuring interoperability.

Open systems. Developing an effective architecture that supports general-purpose, open real-time systems and applications is another challenge facing the real-time community. This architecture would let a dynamic mix of independently developed real-time applications coexist on the same machine or set of machines, possibly embedded in the Internet. Consumers could then run multiple applications with real-time requirements on their general-purpose home and business computers, just as they run nonreal-time applications today.

There are several difficulties associated with an effective real-time architecture supporting open real-time database computing. The architecture must deal with system characteristics that are unknown until runtime, including

- hardware characteristics, such as processor speeds, caches, memory, buses, and I/O devices; and
- the mix of applications and their aggregate resource and timing requirements.

These unknowns make perfect a priori schedulability analysis effectively impossible. Thus, distributed systems will need more flexible approaches than those typically used for building fixed-purpose real-time systems today.

In addition, any architecture must

- obtain real-time performance from legacy databases,
- provide for interoperation between heterogeneous databases with time-sensitive data and transaction deadlines,
- effectively partition data to meet time requirements, and
- create parallel, distributed recovery so the system can interface with the external world as quickly as possible, even before complete recovery occurs.

All of these requirements for distributed open systems present significant research challenges.

Integrating real-time with other properties

As time-critical applications continue to evolve, real-time databases will need to support other properties, including fault tolerance, security, availability, and survivability. These requirements have been studied in isolation, but supporting them together poses unique scientific and engineering challenges.

Supporting combinations of these requirements can be difficult because in some cases they are not compatible, or trade-off strategies are not clearly identified. Consider the integration of security and real-time requirements. Many real-time database applications maintain sensitive information shared by multiple users and applications with different levels of security requirements. In electronic commerce, for example, a real-time database provides critical infrastructure to support complex and flexible services and manage requests with widely varying security requirements in the context of a highly dynamic workload.

Security trade-offs. In general, when resources must be shared dynamically by transactions with different security classes, requirements for real-time performance and security conflict with each other. Frequently, priority inversion is necessary to avoid covert channels (hidden timing channels), which *must* not exist in secure systems.

As an example, consider a high-security, high-priority transaction entering a database, only to find a low-security, low-priority transaction holding a write lock on a data item it needs to access. If the system preempts the low-priority transaction, the principle of noninterference is violated. But if the system delays the high-priority transaction, a priority inversion occurs. Therefore, creating a database that is completely secure and strictly avoids priority inversion is not feasible.

The integration of security and real-time requirements requires some concessions. In some situations, priority inversions might be allowed to protect the security of the system. In other situations, the system might allow covert channels so that transactions can meet their deadlines. However, when a system trades off security, it is only partially secure. In this case, defining the exact meaning of *partial security* is important. A key issue is identifying the correct metrics to evaluate the level of security.

Fault tolerance. There is a similar trade-off when supporting fault tolerance and real-time requirements, as systems can provide different levels of fault-tolerance support to avoid violating timing constraints. For example, depending on the level of fault tolerance, transactions can

- proceed to another node when a processor fails and use replicated data,
- retry themselves in the case of transient faults, or

Other Research Areas

Scaling. Because system timing properties are usually very sensitive to scaling, understanding the complexity introduced by large-scale applications is particularly important in real-time databases.

Query languages and requirements specification. In addition to ongoing efforts to include real-time specifications in Structured Query Language (SQL), specifying real-time requirements in an unambiguous manner and enforcing them consistently needs further study.

Modeling. Various types of data have different kinds of timing properties. Researchers must develop models that associate temporal properties with data types and permit the clear and easy specification of relationships between consistency and timing constraints.

New data formats. As multimedia becomes more pervasive, we need effective methods to support timing requirements for these new data formats.

Benchmarks. Although there are benchmarks for traditional database systems (TPC and 007) and real-time systems (Rhealstones and Hartstones), there are no established benchmarks providing a representative workload for real-time databases.

Integration of active and real-time databases. Real-time systems are inherently reactive—they must respond to external events in the environment as well as internal events triggered by timers or calculated conditions/states. We must develop reactive models that consider time constraints, the formal coupling of events and accompanying responses, and efficient runtime algorithms for detecting events.

Resource management. We must develop and evaluate priority-based scheduling and concurrency control protocols that can, in an integrated and dynamic fashion, manage transactions with precedence, resources (including processor, communication resources, and I/O devices), and timing constraints. In particular, we must integrate resource allocation policies and distributed transaction management protocols.

Operating system internals. Empirical research into the interaction between OSs and real-time database systems is important: We cannot guarantee the correct functioning and timing behavior of real-time database systems without a thorough understanding of the impact of OS internals.

Trade-off analysis. Real-time performance metrics for database correctness, performance, and predictability will let us better understand the trade-offs between satisfying timing constraints and maintaining database consistency. To this end, we need additional methods that enable trade-offs between serializability and timeliness, precision and timeliness, and so forth that will allow us to refine and improve these new metrics. Such performance metrics are especially critical for overload management in real-time database systems.

- produce partial results prior to the deadline to avoid a timing fault.

Users and applications can request different levels of service, depending on the importance of the timing constraints and the system status. One key research issue is mapping user requirements to the underlying database system and object-level requirements. This too is a question of composition. Given the underlying object mechanisms that support fault tolerance, the question is how to compose objects to meet service-level requirements.

Other research issues. Other research challenges exist, too numerous to discuss fully in this article. For completeness' sake, we briefly describe some key issues in the "Other Research Areas" sidebar.

Currently, only a relatively small number of real-time database researchers are tackling these research challenges, and we need to increase activity in this area. Industrial research groups developed the first generation of commercial real-time databases, which typically emerged from telecommunications, manufacturing, and avionics applications where conventional databases did not adequately meet real-time requirements.

With the increasing popularity of audio and video applications, we will likely see more of the open research problems being addressed by large, commercial database companies. We also expect that research issues related to system support and distributed and global systems will be solved first, since they are extensions to today's technology. Research issues involving the integration of real-time and other properties such as security and fault tolerance will likely remain long-term problems due to their difficulty. ❖

References

1. J. Stankovic, "Misconceptions About Real-Time Computing: A Serious Problem For Next Generation Systems," *Computer*, Oct. 1988, pp. 10-19.
2. A. Bestavros, K-J Lin, and S. Son, eds., *Real-Time Database Systems: Issues and Applications*, Kluwer Academic, Boston, 1997.
3. K. Ramamritham, "Real-Time Databases," *J. Distributed and Parallel Databases*, Vol. 1, No. 2, 1993, pp. 199-226.
4. R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Study," *ACM Trans. Database Systems*, Sept. 1992, pp. 513-560.
5. S. Datta et al., "Multiclass Transaction Scheduling and Overload Management in Firm Real-Time Databases," *Information Systems*, Mar. 1996, pp. 29-54.
6. J. Huang et al., "Priority Inheritance in Real-Time Databases," *Real-Time Systems J.*, Sept. 1992.
7. R. Sivasankaran et al., "Priority Assignment in Real-Time Active Databases," *VLDB J.*, Jan. 1996, pp. 19-34.
8. M. Xiong et al., "Scheduling Transactions with Tempo-

ral Constraints: Exploiting Data Semantics," *Proc. Real-Time Systems Symp.*, IEEE Computer Society, Los Alamitos, Calif., 1996, pp. 240-249.
9. S. Chen et al., "Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems," *Real-Time Systems J.*, Sept. 1991, pp. 307-336.
10. G. Ozsoyoglu and R. Snodgrass, "Temporal and Real-Time Databases: A Survey," *IEEE Trans. Knowledge and Data Eng.*, Aug. 1995, pp. 513-532.
11. A. Bestavros and V. Fay-Wolfe, *Real-Time Database and Information Systems: Research Advances*, Kluwer Academic, Boston, 1997.

John A. Stankovic is the BP America Professor and chair of the Computer Science Department at the University of Virginia. His research interests are in distributed computing, real-time systems, operating systems, and distributed multimedia database systems. Stankovic serves on the Board of Directors of the Computer Research Association and has served as the chair of the IEEE Technical Committee on Real-Time Systems. He is the editor-in-chief for IEEE Transactions on Parallel and Distributed Systems. *He is a Fellow of both the IEEE and the ACM and is an IEEE Golden Core Member. Stankovic received a PhD from Brown University.*

Sang Hyuk Son is an associate professor of computer science at the University of Virginia. His research interests include real-time computing, database systems, and information security. He has been working on supporting multidimensional requirements, including real-time, security, and fault tolerance, in distributed object-oriented database systems. He is an associate editor of the IEEE Transactions on Parallel and Distributed Systems *and is the editor of* Advances in Real-Time Systems *(Prentice Hall, 1995), and the co-editor of* Real-Time Database Systems: Issues and Applications *(Kluwer Academic, 1997). Son received a PhD in computer science from the University of Maryland and is a senior member of the IEEE Computer Society.*

Jorgen Hansson is a PhD candidate at the University of Skovde, Sweden, where he received an MSc in computer science. His research interests include active real-time database systems, real-time systems, and scheduling of overloaded systems. He was general co-chair and organizing co-chair for the International Workshop on Active, Real-Time, and Temporal Database Systems, 1995 and 1997, respectively. Hansson was a visiting scholar at the University of Virginia, January to July, 1998.

Contact the authors at the Dept. of Computer Science, University of Virginia, Charlottesville, VA 22903; stankovic@cs.virginia.edu.