Misconceptions About Real-Time Databases

John A. Stankovic, Sang Son, Jorgen Hansson
Department of Computer Science
University of Virginia
Charlottesville, VA 22903

July 22, 1998

Abstract

More and more databases are being used in situations where real-time constraints exist. A set of misconceptions have arisen regarding what a real-time database is and the appropriateness of using conventional databases in real-time applications. Nine misconceptions are identified and discussed. Various research challenges are enumerated and explained. In total, the paper articulates the special nature of real-time databases.

1 Introduction

In 1988 a paper entitled *Misconceptions of Real-Time Computing* was published [11]. This paper articulated the key differences between general purpose computing and real-time computing. The impact of the paper was significant in that it spurred a lot of research that specifically focussed on real-time issues. We believe that there is now a significant body of scientific and technological results in real-time computing, in part, due to the careful definition of real-time computing and the articulation of the important differences between real-time computing and general purpose computing found in that paper.

It is now 10 years later and many computer systems, including general purpose computing systems, need to support real-time properties, e.g., applications with audio and video require real-time support. In addition, the level of data sophistication employed in many real-time systems is growing, ranging from sensor data and various derived data typically found in the more classical real-time systems, to large, even global database management systems that must support real-time characteristics (such as the air traffic control system). Unfortunately, many misconceptions have arisen with respect to the real-time aspects of databases. The state of confusion seems to be similar to what existed in 1988 with respect to the difference between real-time computing and general purpose computing; except now it is with respect to databases. Consequently, in this paper we carefully define real-time databases, state and dispel the most common misconceptions of real-time databases, and encourage increased research into real-time databases.

In the following discussion we are addressing two distinct audiences. The first is the mainstream database researchers and users who usually do not have any experience with real-time issues and often feel that it is acceptable to simply use a commercial database management system and make it go fast! The second audience is the real-time system community who, in the past, have dealt primarily with real-time data from sensors and data derived from this sensor data. Usually, real-time data was in main memory, and when higher level (non-real-time) data was required it was probably placed in a totally separate database system, outside the real-time data purview.

2 Definition

A *real-time database* system is a database system where timing constraints are associated with transactions and data have specific time intervals for which the data are valid [9, 2]. The transaction timing constraints can be deadlines to complete by, times by which they must start, periodic invocations, etc. It is not necessary that every transaction has a timing constraint, only that some transactions do. See Figure 1 for an example of a real-time database application. 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 that indicate that the recorded values are only valid for a certain interval of time. A real-time database makes this validity interval explicit, i.e., it is part of the database schemas. Transaction correctness is then defined as 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 (i.e., within their time validity interval). Logically this means that the value of a data item reflects the true state of

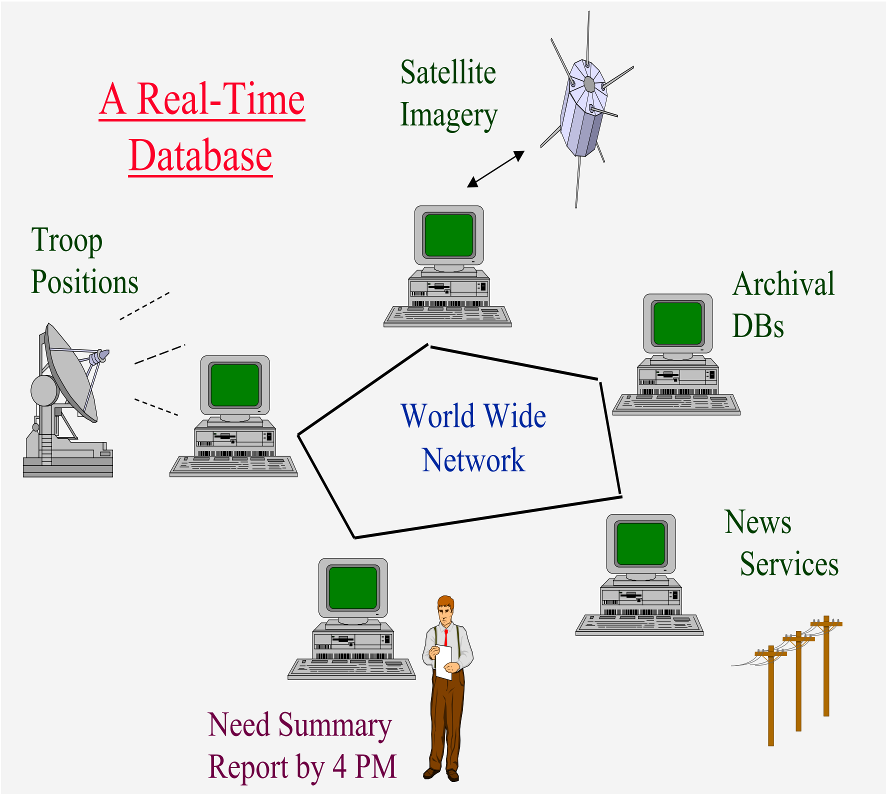

Figure 1: A Real-Time Database Application.

the world to an acceptable degree of accuracy. A relative time consistency constraint among multiple data items used by a transaction means that the times at which those items were updated (sensed) are within a specified time interval of each other. Logically, it means that the states each of the data items represent are within an acceptable time interval of each other. For example, if temperature and pressure are used by a transaction to make some decision regarding a chemical process, that temperature and pressure must correlate closely in time, else the computation would make no sense.

Note also that a system which simply has data with real-time requirements, such as sensor data, does not constitute a real-time database system. Since a real-time database system is a database system, it has queries, schemas, transactions, commit protocols, concurrency control support, and storage management.

3 Misconceptions

To better understand what real-time databases are and how important they are, 9 common misconceptions are enumerated and discussed. The first three misconceptions are variations of the misconstrued theme that real-time relates to speed. The next three are variations of the misconception that current databases can be used. In spite of the similarity of the issues within each of these groups of misconceptions, it is instructive to separate these general themes into these three related aspects.

Advances in hardware will take care of the real-time database requirements.

Technology will exploit parallel processors to improve system throughput, but this does not mean that the timing constraints will be automatically met. In fact, with the increased size and complexity of the database and hardware it will become more and more difficult to meet and show that the timing constraints will be met. Hardware alone cannot ensure that transactions will be scheduled properly to meet their deadlines, nor ensure that the data being used is temporally valid. For example, if a transaction *more quickly* uses out of date data, it is still an incorrect transaction.

Advances in standard database technology will take care of real-time database requirements.

Sometimes database designers claim that better database buffering, faster commit protocols, and novel query processing techniques will speed up databases to permit them to be used in real-time systems. While these techniques will help, they will not ensure that deadlines are met nor support the requirement that transactions only use valid (in time) data. The advances in database technology that will be required include time cognizant protocols for concurrency control, commit processing, transaction scheduling, and logging and recovery. There now exists ample evidence that such protocols are considerably better at supporting real-time transaction and data correctness than standard database protocols which simply *go fast* [1, 6, 7, 10, 12].

Real-time computing is equivalent to fast computing.

The objective of fast computing is to minimize the average response time of a set of transactions. However, the objectives of real-time databases are to meet the timing constraints and the data timing validity requirements of individual transactions and to keep the database time valid via proper update rates. Again, to do this it is necessary to have time cognizant protocols. A simple, contrived example can illustrate this fact. Consider two transactions, A and B. Transaction A starts at time 1 and uses data item X and has a deadline at time 20 and an execution time of 10. Assume that transaction A begins executing at time 1 and locks X at time 2. At time 3 transaction B is invoked and it uses X, has an execution time of 2 and a deadline at time 6. Standard database protocols would have transaction B block (or possibly not even begin executing). Transaction A would then complete and release the lock on X at time 11. However, this is too late for transaction B and B misses its deadline. A time cognizant set of protocols would preempt transaction A at time 3 (because transaction B's deadline is earlier than transaction A) and when B attempts to obtain X, one time cognizant solution (there are others) would be to abort transaction A, let transaction B finish (at time 5) and then restart transaction A. In this case both transactions complete correctly and on-time.

There is no need for a real-time database because we can solve all the problems with current databases.

This is a tricky issue. For example, using a current database system, it is possible to 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. Further, the system can be modified to run some form of earliest deadline scheduling by controlling the priority of each transaction. However, this means that every transaction must program this capability itself instead of having the system support the capability. And, by placing these two features into the system, the designers are, *in fact* moving towards a real-time database system. The problem is that if the transactions have timing constraints and data validity timing constraints then it is more efficient to build this support into the system rather than trying to cajole, manipulate, force fit a current typical database system into this needed set of capabilities. Further, if you actually do this force fitting, then you now have a real-time database system (but it will not likely be as efficient as if you developed these capabilities from the ground up). After all, all algorithms are programmable on a Turing machine, but few people would advocate using a Turing machine to build real systems.

Using a conventional database system and placing the database in main memory is sufficient.

It is sometimes argued that placing a conventional database in main-memory is a viable approach in order to gain performance and thereby make them suitable for real-time systems. Although it is true that main-memory resident databases 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, etc.) that prevent time constraints of transactions from being ensured. Again, increases in performance can not 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 non real-time database designers. Real-time designers often hold the related view, but from a different perspective, i.e., you *must* place the database in main memory. This is not correct either. The primary reasons for placing data in main-memory are speed and to avoid delays introduced by disks, e.g., unpredictable seek and rotational delays. The issue to discuss here is the I/O. In most systems, I/O requests are scheduled in order 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. In the same way traditional CPU scheduling algorithms, attempting to minimize response time or maximize throughput, have been shown to be inappropriate for real-time systems, the use of non-time-cognizant disk scheduling algorithms are not appropriate for scheduling disk requests. It has been shown that disk scheduling algorithms that combine a scan and deadline requirement works considerably better than conventional algorithms [4]. It is likely that some combined solution will prevail where critical data is placed and pinned in (non-volatile) main memory, and other, less critical data is stored on the disk using time cognizant disk scheduling.

A temporal database is a real-time database.

Although both temporal databases and real-time databases support the notion of time, the aspects of time they support are not the same. While a temporal database is aiming to support some aspects of time associated with information (e.g., time-varying information such as stock quotes), a real-time database tries to satisfy timing constraints associated with operational aspects of the database.

Time has several dimensions, but in the context of databases, two dimensions are of particular interest: valid time which denotes the time a fact was true in reality, and transaction time during which the fact was present in the database as stored data [8]. These two dimensions are in general orthogonal, although there could be some application-dependent correlations of the two times. A temporal database identifies those dimensions of time associated with the information maintained in the database and provides support to the user/applications to utilize such timing information, while in real-time databases those time dimensions imply some timing constraints.

Consider the difference between a temporal database and a real-time database via the following example. The military rank of Beetle Bailey can be specified in a temporal database as private during January 1, 1998, through 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. In most real-time databases, such static timing facts are not of primary concern. In a real-time database, the valid time is specified according to the semantics of the counterpart (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 a sensor data was inserted into the database at time T and its valid time is t, then the value must be updated within T+t, and if not updated in 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.

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 properties. However, such approach is, in general, not acceptable in temporal databases. Because of their different objectives, the policies and mechanisms used to resolve data and resource conflicts in real-time databases are different from those 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 hence fairness and resource utilization are secondary considerations.

There is no way to make guarantees or achieve predictability in a real-time database system.

It is sometimes argued that the predictability can not be enforced in a real-time database partly due to the complexity of making accurate and not too pessimistic estimates of the transaction execution times. The complexity rises from the fact that database systems have a number of sources of unpredictability [9]: dependence of the transaction's execution sequence on data values; data and resource conflicts; dynamic paging and I/O; and transaction aborts resulting in rollbacks and restarts. We have already mentioned two solutions to overcome the unpredictability due to disk delays, namely, placing the database in main memory or adopting time-cognizant protocols for scheduling disk requests and managing memory buffers.

While general purpose transactions are difficult to evaluate in terms of their dependence on data values, many real-time transactions are specialized (such as a periodic sensor update transaction), fixed (it uses the same type and amount of data each time), and/or pre-written and evaluated off-line. This set of features enables associated protocols to utilize such information and improve predictability. Due to data conflicts, transactions may be rolled back and restarted, which increases the total execution time of the transaction, and in the worst case, unbounded number of restarts will occur, causing not only the transaction to miss its deadline, but may also jeopardize the timeliness of other transactions requesting resources. Again, in real-time systems the set of transactions are normally well-known and can therefore be pre-analyzed in order to give estimates of the resources required, both in terms of execution time and data that is needed. By scheduling transactions based on this information, conflicts and the number of transaction restarts can then be minimized and bounded. Even though much progress has been made on improving predictability, this issue is still very much an open research question.

A real-time database is a specialized database.

It is true that each real-time database application may have different timing constraints. However, it does not imply that a specialized database system must be developed from scratch for each application. An analogy would be to state that each real-time application needs to develop its own specialized real-time operating system, since its resource requirements and scheduling policies are different from others. Although the specifics of timing requirements can vary among applications, each application need the 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 supporting timing constraints deals with the lowest level database access mechanisms, the overall architecture of database systems needs to be changed to be time cognizant. The reason is similar in explanation as to why certain time-critical applications need real-time operating systems (require fixed priorities, have known worst case latencies, be able to pin pages in memory, etc.) instead of conventional operating systems. However, different real-time operating system don't have to be developed for each application.

4 Research Challenges

While a significant amount of real-time database research has been done [3], this field is still in its early stages. Several key research challenges are now itemized and discussed.

4.1 System Support

In comparison to traditional tasks (processes), database transactions differ in several ways. One area is the differences with respect to obtaining worst case execution times. Obtaining useful worst case execution times for transactions is a complex issue since transactions normally involve multiple resources, such as CPU, I/O, buffers, and data, and the impact of blocking on the transaction response time is also difficult to assess. While the purpose of a concurrency control protocol is to ensure that database consistency is maintained, by controlling the interleaved execution of transactions running concurrently, it also affects the response time of the transaction. Further, the transaction execution time is usually very data dependent, i.e., the execution of a transaction depends on the volume of data and the data values themselves that it reads from the database. Hence, in order to enforce the timeliness of the transactions and maintain database consistency, scheduling algorithms must consider both hardware resources and the scheduling of data resources. Critical issues are obtaining better worst case execution time estimates and determining how concurrency control and the scheduling of transactions should be integrated.

In conventional databases, serializability has been the primary correctness criterion for transaction schedules, i.e., the result produced by the interleaved execution of a set of transactions should be equal to the result produced by executing the transactions in some serial order. Although serializable schedules ensure correctness under a wide range of 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 of utmost importance, in which case it could be advantageous to trade serializability for timeliness, i.e., a non-serializable schedule producing a useful result on time is better than a serializable schedule producing a result too late. A key challenge is to define new and alternative criteria for database correctness, and develop methods which trade serializability for timeliness.

Earlier, the importance of I/O scheduling was discussed. A related and important issue is buffer management. In conventional systems the goal of buffer management is to reduce the transaction response time, and buffer items in these systems are normally allocated and replaced based on the reference behavior of the transactions. The consequence of only considering transaction reference behavior when allocating and replacing buffers in a real-time system is that the performance can be degraded. For example, consider that buffer slots referenced by currently executing (and not yet committed) transactions are replaced, in which case completion of the transactions may be delayed, and in the worst case, deadlines are missed. Other semantics such as periodic transactions must be accounted for in the buffer management policy (a poor policy might have pages replaced just prior to the transaction being reactivated for its next periodic invocation). The research challenge is to develop buffer management policies that consider the importance and the temporal requirements of the transactions and also enforce predictability.

Transaction aborts and transaction recovery are tricky issues in several ways. Performing transaction aborts and recovery consume valuable processing time which may affect other currently executing transactions, and hence, the decision at what time recovery should be performed should be considered cautiously. In systems where lock-based concurrency control protocols are adopted, it is of interest to avoid prolonged waiting for transactions accessing resources locked by the recovering transaction, and hence, locks should be released as early as possible. With optimistic protocols, detection and resolution of data conflicts are both done at the end of the transaction execution, in which case the concurrency control manager is notified and it checks for conflicts. If a conflict is detected, the transaction is aborted and then restarted. A transaction may be restarted a number of times before it can commit successfully. It is not hard to see how this causes a problem in a real-time database system. Not only may the increased processing time due to transaction restarts cause the transaction to become tardy, it may jeopardize the timeliness of other transactions at the same time.

4.2 Distributed and Global Systems

Many real-time systems are built to achieve a specific set of goals and are centralized. The set of tasks to be performed is well understood at system design time and static solutions abound. Large real-time systems typically operate in complex non-deterministic environments where database requirements become important. This gives rise to the need for dynamic solutions and real-time databases. Such applications include, defense systems such as early warning aircraft, command and control, autonomous vehicles, missile (control) systems and complex ship systems, as well as other applications such as air traffic control, the stock market, or video server systems over the Internet. These systems operate for long periods in fault inducing and nondeterministic environments under time constraints. These systems need to be robust while delivering high real-time performance.

Composition has long been recognized as a key research issue for these systems. However, composition has largely focussed on functional composition. A current research objective is to develop the notion of composition across multiple interacting domains: function, time, fault tolerance, and security. Both off-line and on-line solutions are required. The results should permit verification. Thus, the results will lead to adaptive high-performance fault tolerant embedded systems that *dynamically* address real-time constraints, and provide both *a priori* acceptable system-level performance guarantees and graceful degradation in the presence of failures, time constraints, and database access. Any on-line composition is itself subject to time and fault tolerance requirements as well as needing to produce functional, timing and fault tolerant components that create the system actions. How low level real-time techniques interact with real-time database technology is a critical issue.

The distributed nature of the systems gives rise to new database research issues including distributed real-time concurrency control and commit protocols, distributed transaction scheduling, meeting end-to-end timing constraints when database accesses are involved, supporting replication of data in real-time, and interoperability.

Another challenge facing the real-time systems community is how to build and deliver general-purpose, open real-time systems and applications that permit a dynamic mix of multiple independently developed real-time applications to coexist on the same machine or set of machines, possibly embedded in the Internet. Such a real-time architecture would allow consumers to purchase and run multiple applications of their choice, including applications with real-time requirements, on their general-purpose home and business computers, just as they do with non-real-time applications today.

Some of the difficulties of an effective real-time architecture supporting open real-time database computing include:

- Hardware characteristics are unknown until runtime (processor speeds, caches, memory, busses, and I/O devices vary from machine to machine).

- The mix of applications and their aggregate resource and timing requirements are unknown until run-time.

- Perfect *a priori* schedulability analysis is effectively impossible. This means that somewhat different and more flexible approaches will likely be needed than those typically used for building fixed-purpose real-time systems today.

- How to get real-time performance out of legacy databases.

- How to interoperate between heterogeneous databases with time sensitive data and where transactions have deadlines.

- How to partition data to meet time requirements.

- How to create parallel, distributed recovery so that the system can interface to the external world as quickly as possible, even before the complete recovery occurs.

4.3 Integration of Real-Time with Other Properties

As time-critical applications continue to evolve, real-time databases become more complex and need to support other properties, in addition to real-time requirements. Those requirements include fault-tolerance, security, availability, and survivability. Those requirements have been studied in isolation, but the need for supporting them together poses scientific and engineering challenges that should be tackled to develop practical solutions for advanced real-time database systems.

The difficulty in supporting combinations of requirements stems from the fact that in many cases, those requirements are not compatible with each other or the tradeoff strategies are not clearly identified. Consider the integration of security and real-time requirements. In many real-time database applications, security is an important requirement, since the system maintains sensitive information to be shared by multiple users and applications with different levels of security requirements. For example, electronic commerce is an interesting application where both security and real-time requirements should be considered together. In electronic commerce, a real-time database is a critical infrastructure that is essential to support complex and flexible services to manage requests in the context of a highly dynamic workload with widely varying requirements. The problem is that in general, when resources must be shared dynamically by transactions with different security classes, requirements of real-time performance and security conflict with each other. Frequently, priority inversion is necessary to avoid covert channels (which are hidden timing channels that must not exist for secure systems).

Consider a transaction with a high security level and a high priority entering the database, and it finds that a transaction with a lower security level and a lower priority holds a write lock on a data item that it needs to access. If the system preempts the lower priority transaction to allow the higher priority transaction to execute, the principle of non-interference is violated, for the presence of a high security transaction affects the execution of a lower security transaction. On the other hand, 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. A system that wishes to accomplish the integration of security and real-time requirements must make some concessions at times. 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. When the system has to trade-off security, the system is no longer completely secure; rather it only will be partially secure. In that case, it is extremely important to define the exact meaning of partial security. A key issue is identifying the correct metrics to evaluate the level of security obtained.

Supporting fault-tolerance and real-time requirements has a similar trade-off problem as in supporting security and real-time requirements. Different levels of fault-tolerance support can be provided to avoid violating timing constraints. For example, a transaction is allowed to proceed with other nodes in which the data is replicated when a processor fails, be retried if transient faults occur, and can produce partial results prior to the deadline to avoid a timing fault. Users/applications can request different levels of service, depending on the importance of the timing constraints and the system status. One key research issue to be investigated is mapping the user level requirements to the underlying database system and object level requirements. This research question is again one of composition. That is, given the underlying object mechanisms that support fault tolerance how can objects be composed to meet the service level requirements.

4.4 Other Research Areas

Many other research challenges exist: too numerous to discuss fully in this paper. Some of these are briefly discussed below.

- Scaling: Understanding the complications introduced by the complexity of a large-scale applications. This is particularly serious in real-time databases, since the timing properties of the system is usually sensitive to scaling factors.

- Query languages and requirements specification: Although there is an on-going effort to include real-time specification in SQL, how to specify real-time requirements in an unambiguous manner and how to enforce them need further study.

- Modeling: Given the different kinds of timing properties of the various types of data, the data types must be modeled so that temporal properties can be associated with the data. Relationships between consistency constraints and timing constraints need to be easily and clearly specified.

- New data formats: As more data is stored and manipulated using multimedia, effective methods to support timing requirements to handle them are necessary.

- Benchmarks: A good benchmark should provide representative workload to evaluate important real-time capabilities of the system. While several benchmarks have been used for traditional database systems (e.g., TPC and 007 benchmarks) and real-time systems (e.g., Rhealstones and Hartstones benchmarks) benchmarks are needed for real-time databases.

- Integration of active and real-time databases: Since real-time systems are inherently reactive, they must respond to external events occurring in the environment as well as internal events triggered by timers or calculated conditions/states. This requires the development of reactive models that consider time constraints, formal reasoning of coupling between events occurring and actions executed as responses, and efficient run-time algorithms for detecting events.

- Resource management: Development and evaluation of priority-based scheduling protocols 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, resource allocation policies and distributed transaction management protocols must be integrated.

- Empirical research: The interaction between OS and real-time database system can be best understood through empirical research. It is important since the correct functioning and timing behavior of real-time database systems cannot be guaranteed without a thorough understanding of the impact of OS internals.

- Trade-off analysis: It is important to understand the possible trade-offs between satisfying timing constraints and maintaining database consistency, and develop metrics for database correctness, performance, and predictability. Methods that enable the trade-offs between serializability and timeliness, between precision and timeliness, and other types of trade-offs that can be used to improve the new real-time performance metrics need to be studied. They are especially critical for overload management in real-time database systems.

5 Conclusions

This paper defines real-time databases as well as two key aspects of real-time databases: absolute and relative timing consistency. In these databases both transactions **AND** data have timing constraints. These databases are often involved with sensors and actuators (including audio and video). Using a discussion of 9 common misconceptions and a list of open research challenges, the special nature of real-time databases is articulated. More research is required and the field is becoming more and more important as enterprise systems of all types are utilizing sophisticated database information with real-time requirements.

References

[1] R. Abbott and H. Garci-Molina, Scheduling Real-Time Transactions: A Performance Study, *ACM Transactions on Database Systems*, 17(3):513-560, September 1992.

[2] A. Bestavros, K-J Lin, and S. Son, editors, *Real-Time Database Systems: Issues and Applications*, Kluwer Academic Publishers, Boston, 1997.

[3] A. Bestavros and V. Fay-Wolfe, *Real-Time Database and Information Systems: Research Advances*, Kluwer Academic Publishers, Boston, 1997.

[4] S. Chen, J. Stankovic, J. Kurose, and D. Towsley, Performance Evaluation of Two New Disk Scheduling Algorithms for Real-Time Systems, *Real-Time Systems Journal*, Vol. 3, No. 3, September 1991.

[5] M.J. Carey, R. Jauhari, and M. Livny, Priority in DBMS Resource Scheduling, *Proceedings of the 15th VLDB Conference*, August 1989.

[6] A. Datta, S. Mukherjee, P. Konana, I. Viguier, and A. Bajaj, Multiclass Transaction Scheduling and Overload Management in Firm Real-Time Databases, *Information Systems*, Vol. 21, No. 1, pp. 29-54, March 1996.

[7] J. Huang, J. 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.

[8] G. Ozsoyoglu and R. Snodgrass, Temporal and Real-Time Databases: A Survey, *IEEE Transactions on Knowledge and Data Engineering*, 7(4), pp 513-532, August 1995.

[9] K. Ramamritham, Real-Time Databases, *Journal of Distributed and Parallel Databases*, Vol. 1, No. 2, pp. 199-226, 1993.

[10] R. Sivasankaran, J. Stankovic, D. Towsley, B. Purimetla, and K. Ramamritham, Priority Assignment in Real-Time Active Databases, *VLDB Journal*, Vol. 5, No. 1, pp. 19-34, January 1996.

[11] J. Stankovic, Misconceptions About Real–Time Computing: A Serious Problem For Next Generation Systems, *IEEE Computer*, Vol. 21, No. 10, pp. 10-19, October 1988.

[12] M. Xiong, R. Sivasankaran, J. Stankovic, K. Ramamritham and D. Towsley, Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics, *Real-Time Systems Symposium*, December 1996.