Real-Time Database — Similarity Semantics and Resource Scheduling

Tei-Wei Kuo*
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan 621
R.O.C.

Aloysius K. Mok†
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
U.S.A.

Abstract

While much recent work has focussed on the performance of transaction systems where individual transactions have deadlines, our research addresses the semantics of data usage in real-time applications and its integration with real-time resource management, in particular, the timeliness value of real-time data and the inherent path and not state-based constraints on concurrency control. Central to our research is the idea of similarity which is a reflexive, symmetric relation over the domain of a data object. By exploiting the similarity relation, we propose a class of efficient data-access policies for real-time data objects. We shall also discuss the design of a distributed real-time data-access interface. Our goal is to build a database facility which can support predictable real-time applications involving high-speed communication, information access, and multimedia.

1 Introduction

In recent years, a number of researchers have studied the problem of meeting the timing requirements of applications while preserving database consistency. However, most of extant work has focussed on the performance of transaction systems to meet individual transaction deadlines. Overly strict criteria such as serializability are often used to ensure the correctness of such applications. Such a brute-force approach may commit a significant amount of resources for no real gains.

*Supported in part by a research grant from the National Science Council under Grant NSC85-2213-E-194-008

†Supported in part by a research grant from the Office of Naval Research under ONR contract number N00014-89-J-1472

This paper examines the time-volatility of real-time databases which set them apart from conventional databases and provides new correctness criteria for concurrency control which are germane to real-time applications. We discuss some efficient real-time data-access policies which exploit a *similarity* relation over real-time data objects. Based on these ideas, we have also designed a distributed real-time data-access interface which integrates real-time resource management and object-oriented design methodologies.

The rest of the paper is organized as follows. Section 2 describes our real-time data model. Section 3 explores a weaker correctness criterion for concurrency control in real-time transactions, by investigating the notion of *similarity*. Section 4 proposes a class of real-time data-access protocols called SSP (Similarity Stack Protocols) based on the concept of similarity. Section 5 describes a distributed real-time data access package which can be run on a multiprocessor platform. Section 6 is the conclusion.

2 A Real-Time Database Model

2.1 Real-Time Data Objects

A real-time database is a collection of data objects which are used to model a time-critical dynamic system in the real world. Each data object takes its value from its *domain*. We define a database state as an element of the Cartesian product of the domains[10] of its data objects. A database state may be represented by a vector of data values such that every data object is a component of this vector.

Because of the dynamic nature of the real world, the useful lifespan of real-time data is usually relatively short. For the purpose of measuring the timeliness of real-time data, we assign each real-time data object a parameter, *age* which measures the recency of its value. In general, the age of a data object is given by an application-defined procedure which assigns timestamps to its values. Whether the age of a data object is up-to-date may depend on two or more timing constraints in the application. For example, suppose the value of a data object x depends on a data object y, the update transaction of x never misses its deadline, but the update transaction of y often misses its deadline. Then we cannot say that the x value is up-to-date simply because the transaction updating x is always timely.

2.2 Events, Transactions, and Schedules

Events are primitive database read or write operations. A *transaction* is the template of its instances; a transaction instance is a partial order of events. An instance of a transaction is scheduled for every request of the transaction. An *interpretation* of a set of transactions is a collection of transaction definitions and data domain definitions[10].

A *schedule* for a set of transactions is a partial order of events issued by instances of the transaction set. Each event in a schedule is issued by one transaction instance. The ordering of events in a schedule must be consistent with the event ordering as specified by the transaction set. A *serial schedule* is a sequence of transaction instances, i.e., a schedule in which the transaction instances are totally ordered.

A real-time computation may be represented as a collection of events with time stamps. The time stamp of an event in the computation indicates the time it occurs. Events with such time stamps are *timed events*; a real-time computation is a collection of timed events. Let a *timed schedule* for a set of transactions be a collection of timed events issued by instances of the transaction set. It is clear that corresponding to each timed schedule is a unique, untimed schedule which preserves the time stamp order of events in the timed schedule.

3 Correctness Criteria

In this section, we propose a weaker correctness criterion for concurrency control in real-time transactions, by introducing the notion of *similarity*[3].

3.1 Data Similarity

A similarity relation is a binary relation on the domain of a data object. Every similarity relation is reflexive and symmetric, but not necessarily transitive. Different transactions may induce different similarity relations on the same data object domain. *Two views of a transaction are similar* iff every read event in both views uses similar values with respect to the transaction. We say that *two values of a data object are similar* if all transactions which may read them consider them as similar.

In a schedule, we say that two *event instances are similar* if they are of the same type and access similar values of the same data object. We say that *two database states are similar* if the corresponding values of every data object in the two states are similar.

A minimal restriction on the similarity relation that makes it interesting for concurrency control is the requirement that it is preserved by every transaction, i.e., if a transaction T maps database state s to state t and state s' to t', then t and t' are similar if s and s' are similar. We say that a similarity relation is *regular* if it is preserved by all transactions. From now on, we shall be concerned with regular similarity relations only. Further restrictions on the similarity predicate will yield a correctness criterion for transaction scheduling that can be checked efficiently.

3.2 View Δ-Serializability

Our proposed criteria can be viewed as extensions of the standard serializability-based correctness criteria[10] to exploit the concept of similarity. Three correctness criteria defined in [3] are final-state, view and conflict Δ-serializability. Other different correctness criteria have been proposed for different purposes and application areas[2, 8, 11]. Because of space limitation, only the definition of view Δ-serializability is included in this paper.

The transaction view of a transaction instance is a vector of data object values such that the i_{th} component is the value read by the i_{th} read event of the transaction instance[10].

Definition 1 [3] **View Similar:**
A schedule is *view-similar* to another schedule iff

1. They are over the same set of transactions (transaction instances).

2. For any initial state and under any interpretation, they transform similar initial database

states into similar database states with respect to their transaction sets, respectively.

3. Every transaction instance has similar views in both schedules for any initial state and under any interpretation.

It is clear that, if a schedule is view-equivalent to another schedule, then it is view-similar to that schedule, but the converse may not hold. Note that the view-similarity relation between schedules is reflexive and symmetric but not necessarily transitive. A schedule is view Δ-serializable iff it is view-similar to a serial schedule.

Example 1 *view similarity and view Δ- serializability*

Consider the following two schedules π_1 and π_2 in which events are listed in their order of occurrence. (The symbol $\rightarrow$ denotes "to be continued on the next line".) Events $R(\tau_{i,j}, X)$ and $W(\tau_{i,j}, X)$ are read and write operations issued by transaction instance $\tau_{i,j}$ on data object X, respectively.

A schedule

$$\begin{aligned}\pi_1 = \;& W(\tau_{3,1}, X), R(\tau_{1,1}, X), W(\tau_{1,1}, X), R(\tau_{2,1}, X), \rightarrow \\ & R(\tau_{2,1}, Y), W(\tau_{2,1}, Y), W(\tau_{1,1}, Y)\end{aligned}$$

is view similar to another schedule

$$\begin{aligned}\pi_2 = \;& \tau_{3,1}, \tau_{2,1}, \tau_{1,1} \\ = \;& W(\tau_{3,1}, X), R(\tau_{2,1}, X), R(\tau_{2,1}, Y), \rightarrow \\ & W(\tau_{2,1}, Y), R(\tau_{1,1}, X), W(\tau_{1,1}, X), W(\tau_{1,1}, Y)\end{aligned}$$

if $W(\tau_{3,1}, X)$ and $W(\tau_{1,1}, X)$ are similar. Since π_2 is a serial schedule, π_1 is view Δ-serializable. $\Box$

Theorem 1 [3] *The problem of deciding whether a schedule is view Δ-serializable is NP-Hard.*

4 Similarity-Based Concurrency Control

The idea of similarity is certainly not new in practice. In avionic systems, the dynamics of a sensor or the environment may impose an upper bound on the variation of a sensor value over a short time interval. For certain computations, avionic engineers often consider the change in sensor reading over a few consecutive cycles to be insignificant. It is sometimes acceptable to use a sensor value that is not the most recent update in a transaction. *Our contribution is to provide a justification for this ad hoc engineering practice.* More importantly, the similarity relation provides a formal interface for the application engineer to capture the real-time characteristics of his data, so that concurrency control theory can be applied. In the following we describe a class of scheduling policies based on the concept of similarity to provide application engineers more flexibility in concurrency control.

4.1 Strong Similarity

Our definition of regular similarity only requires a similarity relation to be preserved by every transaction, so that the input value of a transaction can be swapped with another in a schedule if the two values are related by a regular similarity relation. Unless a similarity relation is also transitive, it is in general incorrect to swap events an arbitrary number of times in a schedule. For example, let v_1, v_2, v_3 be three values of a data object such that v_1 and v_2 are similar, as are v_2 and v_3. A transaction instance reading v_1 as input will produce similar output as one that reads v_2 as input. Likewise, the same transaction reading v_2 as input will produce similar output as one that reads v_3 as input. However, there is no guarantee that the output of the transaction reading v_1 as input will be similar to one reading v_3 as input, since v_1 and v_3 may not be related under the regular similarity relation. Swapping events two or more times may result in a transaction reading a value that is not similar to the input value before event swapping, and is hence unacceptable. To overcome this problem, we add another restriction to the similarity relation such that swapping similar events in a schedule will always preserve similarity in the output.

This restriction is motivated by the observation that the state information of many real-time systems is "volatile", i.e., these systems are designed in such a way that system state is determined completely by the history of the recent past, e.g., the velocity and acceleration of a vehicle are computed from the last several values of the vehicle's position from the position sensor. Unless events in a schedule may be swapped in such a way that a transaction reads a value that is derived from the composition of a long chain of transactions that extends way into the past, a suitable similarity relation may be chosen such that output similarity is preserved by limiting the "distance" between inputs that may be read by a transaction before and after swapping similar events in a schedule. Thus if two events in a schedule are strongly similar (i.e., they are either both writes or both reads, and the two data values involved are strongly similar), then they can always be swapped in a schedule without violating data consistency requirements. Strong similarity is established by the "write-length" of the data dependency graph, which in turn can be related to update frequencies in practice.

4.2 Similarity Stack Protocols (SSP)

We assume that the application semantics allows us to derive a *similarity bound* for each data object such that two write events on the data object must be strongly similar if their time-stamps differ by an amount no greater than the similarity bound.

The basic strategy of the SSP protocols can be summarized as follows: Transactions are normally scheduled according to their priorities which can be dynamic (e.g., earliest-deadline-first) or static (e.g., as determined by the rate monotonic assignment)[9], with the provision that transaction execution follows the stack discipline, i.e., if transaction B starts after transaction A, then A cannot resume until after B finishes. However, no transaction is allowed to start execution if it conflicts with another transaction which has already started but not committed such that the conflicting read/write events may not be arbitrarily swapped under the similarity relation in the following way:

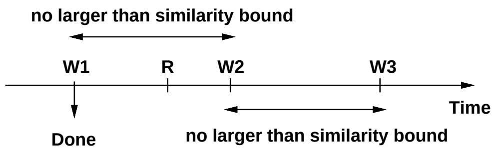

Figure 1: Similarity of conflicting events

Suppose two events e_1 and e_2 conflict with each other. Let e_1 and e_2 be the write events w_2 and w_3, respectively. If their write values are similar under the similarity bound as shown in Figure 1, these two events are similar and it does not matter the result of which write is read by subsequent read events. Suppose e_1 and e_2 are respectively, the write event w_2 and the read event r in Figure 1. For their relative ordering to be unimportant, there must exist an earlier write event whose write value is similar to the write value of w_2 under the similarity bound. If this is the case, as is shown in Figure 1, then it does not matter which write value the read event r reads. The same argument applies to the case where e_1 and e_2 are a read event and a write event, respectively.

It can be shown that the SSP protocols are deadlock-free, subject to limited blocking and satisfy view Δ-serializability. This class of protocols offer better performance, especially on multiprocessor systems, for managing real-time data access. Intuitively, this is what one would expect since many lock-based protocols tend to artificially serialize computation whereas SSP uses no locks at all! We refer readers to [4, 5, 6] for details.

5 Real-Time Object Management Interface (RTOMI)

We have designed RTOMI (Real-Time Object Management Interface) to evaluate algorithms for process scheduling and data distribution based on the similarity concept[7]. RTOMI supports various data-access services ranging from data acquisition to high-leveled data-access commands.[1] It provides the necessary system calls, manages data across processors, and schedules transactions preemptively by various real-time resource-scheduling techniques. Although RTOMI is currently being implemented only on an Intel multiboard computer, its design is entirely architecture-independent.

5.1 Objects Attributes and Coherence

Data objects in RTOMI are virtually/physically continuous blocks of memory space. They may be associated with a set of user-defined procedures that detect violations of the consistency constraints which can be internal, temporal, external, or dynamic[3, 6]. By hiding implementation details, RTOMI not only provides a clean and efficient interface for data accesse but also avoids complicated object declarations and the usual enormous number of associated data-manipulation routines.

Real-time data objects in RTOMI can be duplicated and distributed over network hosts. Data coherence is justified by the similarity relation. For traditional non-real-time data objects, the similarity relation degenerates into the "identity" relation.

5.2 Concurrency Control

Users may use real-time resource scheduling algorithms provided by RTOMI which consider both data synchronization and real-time process scheduling. RTOMI allows users to program their data synchronization protocols (through *rtomi_lock()* and

[1] Providing the *best* data-access services depends on the context of individual applications. RTOMI aims at identifying "essential" data-access services in distributed real-time database systems.

rtomi_unlock() system calls), such as the two-phase locking protocol, in a process code. For now, only independent real-time resource scheduling algorithms such as earliest-deadline-first or fixed priority[9] are allowed. Otherwise, conflicts between data synchronization protocols and other real-time resource scheduling algorithms in handling data synchronization may leave the system in an undefined state.

5.3 Read and Write

RTOMI not only supports primitive data-access operations such as read or write but also provides some advanced data-access operations such as "periodic-read" requests. Note that many real-time applications such as a Video-On-Demand system need to provide "periodic" presentation services for continuous media streams. The ordinary read and write operations are called *value reads* and *value writes*, respectively. The advanced read and write operations are called *action reads* or *action writes*.

6 Summary

Our research provides a framework for understanding and exploiting an important aspect of real-time applications. We have introduced the concept of similarity which has been used on an ad hoc basis by application engineers to provide more flexibility in concurrency control. We have proposed weaker consistency requirements and a class of real-time, data-access protocols, all based on the similarity concept. We have also obtained encouraging experimental performance results. We have also proposed a real-time object management interface to provide low-level mechanisms to facilitate the implementation of real-time, data-intensive applications.

Tailoring a real-time database system to cater to the needs of a wide variety of applications and to achieve good utilization of resources is very important in many real-time applications. An interesting direction of this work is to characterize the domain-specific similarity relation of data in applications such as multimedia, real-time knowledge base, advanced communication and control systems, and to provide a facility for application engineers to specify similarity relations for complex objects. The architecture of the real-time database can then be optimized for efficient similarity-based data access.

References

[1] S. B. Davidson and A. Watters, "Partial Computation in Real-Time Database Systems," *IEEE 5th Workshop on Real-time Software and Operating Systems*, May 1988.

[2] H. Garcia-Molina and G. Wiederhold, "Read-Only Transactions in a Distributed Database," *ACM Transactions on Database Systems*, Vol. 7, No. 2, June 1982.

[3] Tei-Wei Kuo and Aloysius K. Mok, "Application Semantics and Concurrency Control of Real-Time Data-Intensive Applications," *IEEE 13th Real-Time Systems Symposium*, 1992.

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

[5] Tei-Wei Kuo and Aloysius K. Mok, "Using Data Similarity to Achieve Synchronization for Free," *IEEE 11th Workshop on Real-Time Operating Systems and Software*, May 1994.

[6] Tei-Wei Kuo, "Real-Time Database — Semantics and Resource Scheduling," Ph.D. dissertation, University of Texas at Austin, 1994.

[7] Tei-Wei Kuo and Aloysius K. Mok, "The Design and Implementation of A Real-Time Object Management Interface," *1995 IEEE Real-Time Technology and Applications Symposium*, May 1995.

[8] H. F. Korth and G. D. Speegle, "Formal Model of Correctness Without Serializability," *In Proceedings of 1988 ACM SIGMOD Conference on Management of Data*, 1988.

[9] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a hard Real-Time Environment," *Journal of the ACM,* Vol. 20, No. 1, January 1973.

[10] C. Papadimitriou, "The Theory of Database Concurrency Control," *Computer Science Press*, 1986.

[11] K. Ramamritham and C. Pu, "A Formal Characterization of Epsilon Serializability," *Technical Report*, CUCS-044-91, Dept. of Computer Science, Columbia University, 1991.

[12] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *Technical Report* CMU-CS-87-181, Dept. of Computer Science, CMU, November, 1987. *IEEE Transactions on Computers*, Vol. 39, No. 9, September 1990.